

NOAA Technical Memorandum NWS WR-192



NWR VOICE SYNTHESIS PROJECT: PHASE I

Salt Lake City, Utah
January 1986

**U.S. DEPARTMENT OF
COMMERCE**

National Oceanic and
Atmospheric Administration

National Weather
Service



NOAA TECHNICAL MEMORANDA
National Weather Service, Western Region Subseries

The National Weather Service (NWS) Western Region (WR) Subseries provides an informal medium for the documentation and quick dissemination of results not appropriate, or not yet ready, for formal publication. The series is used to report on work in progress, to describe technical procedures and practices, or to relate progress to a limited audience. These Technical Memoranda will report on investigations devoted primarily to regional and local problems of interest mainly to personnel, and hence will not be widely distributed.

Papers 1 to 25 are in the former series, ESSA Technical Memoranda, Western Region Technical Memoranda (WRTM); papers 24 to 59 are in the former series, ESSA Technical Memoranda, Weather Bureau Technical Memoranda (WBTM). Beginning with 60, the papers are part of the series, NOAA Technical Memoranda NWS. Out-of-print memoranda are not listed.

Papers 2 to 22, except for 5 (revised edition), are available from the National Weather Service Western Region, Scientific Services Division, P. O. Box 11188, Federal Building, 125 South State Street, Salt Lake City, Utah 84147. Paper 5 (revised edition), and all others beginning with 25 are available from the National Technical Information Service, U. S. Department of Commerce, Sillis Building, 5285 Port Royal Road, Springfield, Virginia 22161. Prices vary for all paper copy; \$3.50 microfiche. Order by accession number shown in parentheses at end of each entry.

ESSA Technical Memoranda (WRTM)

- 2 Climatological Precipitation Probabilities. Compiled by Lucianne Miller, December 1965.
- 3 Western Region Pre- and Post-FP-3 Program, December 1, 1965, to February 20, 1966. Edward D. Diemer, March 1966.
- 5 Station Descriptions of Local Effects on Synoptic Weather Patterns. Philip Williams, Jr., April 1966 (revised November 1967, October 1969). (PB-17800)
- 8 Interpreting the RAREP. Herbert P. Benner, May 1966 (revised January 1967).
- 11 Some Electrical Processes in the Atmosphere. J. Latham, June 1966.
- 17 A Digitalized Summary of Radar Echoes within 100 Miles of Sacramento, California. J. A. Youngberg and L. S. Overaas, December 1966.
- 21 An Objective Aid for Forecasting the End of East Winds in the Columbia Gorge, July through October. D. John Coparans, April 1967.
- 22 Derivation of Radar Horizons in Mountainous Terrain. Roger G. Pappas, April 1967.

ESSA Technical Memoranda, Weather Bureau Technical Memoranda (WBTM)

- 25 Verification of Operational Probability of Precipitation Forecasts, April 1966-March 1967. W. W. Dickey, October 1967. (PB-176240)
- 26 A Study of Winds in the Lake Mead Recreation Area. R. P. Augulis, January 1968. (PB-177830)
- 28 Weather Extremes. R. J. Schmidl, April 1968. **(Revised December 1983)**
- 29 Small-Scale Analysis and Prediction. Philip Williams, Jr., May 1968. (PB-178425)
- 30 Numerical Weather Prediction and Synoptic Meteorology. Capt. Thomas D. Murphy, U.S.A.F., May 1968. (AD-673365)
- 31 Precipitation Detection Probabilities by Salt Lake ARTC Radars. Robert K. Belesky, July 1968. (PB-179084)
- 32 Probability Forecasting--A Problem Analysis with Reference to the Portland Fire Weather District. Harold S. Ayer, July 1968. (PB-179289)
- 35 Joint ESSA/FAA ARTC Radar Weather Surveillance Program. Herbert P. Benner and DeVon B. Smith, December 1968 (revised June 1970). AD-681857)
- 36 Temperature Trends in Sacramento--Another Heat Island. Anthony D. Lentini, February 1969. (PB-183055)
- 37 Disposal of Logging Residues without Damage to Air Quality. Owen P. Cramer, March 1969. (PB-183057)
- 39 Upper-Air Lows over Northwestern United States. A. L. Jacobson, April 1969. (PB-184296)
- 40 The Man-Machine Mix in Applied Weather Forecasting in the 1970's. L. W. Snellman, August 1969. (PB-185068)
- 42 Analysis of the Southern California Santa Ana of January 15-17, 1966. Barry B. Aronovitch, August 1969. (PB-185670)
- 43 Forecasting Maximum Temperatures at Helena, Montana. David E. Olsen, October 1969. (PB-185762)
- 44 Estimated Return Periods for Short-Duration Precipitation in Arizona. Paul C. Kangieser, October 1969. (PB-187763)
- 46 Applications of the Net Radiometer to Short-Range Fog and Stratus Forecasting at Eugene, Oregon. L. Yee and E. Bates, December 1969. (PB-190476)
- 47 Statistical Analysis as a Flood Routing Tool. Robert J. C. Burnash, December 1969. (PB-188744)
- 48 Tsunami. Richard P. Augulis, February 1970. (PB-190157)
- 49 Predicting Precipitation Type. Robert J. C. Burnash and Floyd E. Hug, March 1970. (PB-190962)
- 50 Statistical Report on Aeroallergens (Pollens and Molds) Fort Huachuca, Arizona, 1969. Wayne S. Johnson, April 1970. (PB-191743)
- 51 Western Region Sea State and Surf Forecaster's Manual. Gordon C. Shields and Gerald B. Burdwell, July 1970. (PB-193102)
- 52 Sacramento Weather Radar Climatology. R. G. Pappas and C. M. Veillette, July 1970. (PB-193347)
- 54 A Refinement of the Vorticity Field to Delineate Areas of Significant Precipitation. Barry B. Aronovitch, August 1970. (PB-194394)
- 55 Application of the SSARR Model to a Basin without Discharge Record. Vail Schermerhorn and Donal W. Kuehl, August 1970. (PB-194389)
- 56 Areal Coverage of Precipitation in Northwestern Utah. Philip Williams, Jr., and Werner J. Heck, September 1970. (PB-194389)
- 57 Preliminary Report on Agricultural Field Burning vs. Atmospheric Visibility in the Willamette Valley of Oregon. Earl M. Bates and David O. Chilcote, September 1970. (PB-194710)
- 58 Air Pollution by Jet Aircraft at Seattle-Tacoma Airport. Wallace R. Donaldson, October 1970. (COM-71-00017)
- 59 Application of PE Model Forecast Parameters to Local-Area Forecasting. Leonard W. Snellman, October 1970. (COM-71-00016)

NOAA Technical Memoranda (NWS WR)

- 60 An Aid for Forecasting the Minimum Temperature at Medford, Oregon. Arthur W. Fritz, October 1970. (COM-71-00120)
- 63 700-mb Warm Air Advection as a Forecasting Tool for Montana and Northern Idaho. Norris E. Woerner, February 1971. (COM-71-00349)
- 64 Wind and Weather Regimes at Great Falls, Montana. Warren B. Price, March 1971.
- 66 A Preliminary Report on Correlation of ARTCC Radar Echoes and Precipitation. Wilbur K. Hall, June 1971. (COM-71-00829)
- 69 National Weather Service Support to Soaring Activities. Ellis Burton, August 1971. (COM-71-00956)
- 71 Western Region Synoptic Analysis-Problems and Methods. Philip Williams, Jr., February 1972. (COM-72-10433)
- 74 Thunderstorms and Hail Days Probabilities in Nevada. Clarence M. Sakamoto, April 1972. (COM-72-10554)
- 75 A Study of the Low Level Jet Stream of the San Joaquin Valley. Ronald A. Willis and Philip Williams, Jr., May 1972. (COM-72-10707)
- 76 Monthly Climatological Charts of the Behavior of Fog and Low Stratus at Los Angeles International Airport. Donald M. Gates, July 1972. (COM-72-11140)
- 77 A Study of Radar Echo Distribution in Arizona During July and August. John E. Hales, Jr., July 1972. (COM-72-11136)
- 78 Forecasting Precipitation at Bakersfield, California, Using Pressure Gradient Vectors. Earl T. Riddiough, July 1972. (COM-72-11146)
- 79 Climate of Stockton, California. Robert C. Nelson, July 1972. (COM-72-10920)
- 80 Estimation of Number of Days Above or Below Selected Temperatures. Clarence M. Sakamoto, October 1972. (COM-72-10021)
- 81 An Aid for Forecasting Summer Maximum Temperatures at Seattle, Washington. Edgar G. Johnson, November 1972. (COM-73-10150)
- 82 Flash Flood Forecasting and Warning Program in the Western Region. Philip Williams, Jr., Chester L. Glenn, and Roland L. Raetz, December 1972, (revised March 1978). (COM-73-10251)
- 83 A Comparison of Manual and Semiautomatic Methods of Digitizing Analog Wind Records. Glenn E. Rasch, March 1973. (COM-73-10669)
- 86 Conditional Probabilities for Sequences of Wet Days at Phoenix, Arizona. Paul C. Kangieser, June 1973. (COM-73-11264)
- 87 A Refinement of the Use of K-Values in Forecasting Thunderstorms in Washington and Oregon. Robert Y. G. Lee, June 1973. (COM-73-11276)
- 89 Objective Forecast Precipitation over the Western Region of the United States. Julia N. Paegle and Larry P. Kierulff, Sept. 1973. (COM-73-11946/3AS)
- 91 Arizona "Eddy" Tornadoes. Robert S. Ingram, October 1973. (COM-73-10465)
- 92 Smoke Management in the Willamette Valley. Earl M. Bates, May 1974. (COM-74-11277/AS)
- 93 An Operational Evaluation of 500-mb Type Regression Equations. Alexander E. MacDonald, June 1974. (COM-74-11407/AS)
- 94 Conditional Probability of Visibility Less than One-Half Mile in Radiation Fog at Fresno, California. John D. Thomas, August 1974. (COM-74-11555/AS)
- 96 Map Type Precipitation Probabilities for the Western Region. Glenn E. Rasch and Alexander E. MacDonald, February 1975. (COM-75-10428/AS)
- 97 Eastern Pacific Cut-Off Low of April 21-28, 1974. William J. Alder and George R. Miller, January 1976. (PB-250-711/AS)
- 98 Study on a Significant Precipitation Episode in Western United States. Ira S. Brenner, April 1976. (COM-75-10719/AS)
- 99 A Study of Flash Flood Susceptibility--A Basin in Southern Arizona. Gerald Williams, August 1975. (COM-75-11360/AS)
- 102 A Set of Rules for Forecasting Temperatures in Napa and Sonoma Counties. Wesley L. Tuft, October 1975. (PB-246-902/AS)
- 103 Application of the National Weather Service Flash-Flood Program in the Western Region. Gerald Williams, January 1976. (PB-253-053/AS)
- 104 Objective Aids for Forecasting Minimum Temperatures at Reno, Nevada, During the Summer Months. Christopher D. Hill, January 1976. (PB-252-866/AS)
- 105 Forecasting the Mono Wind. Charles P. Ruscha, Jr., February 1976. (PB-254-650)
- 106 Use of MOS Forecast Parameters in Temperature Forecasting. John C. Plankinton, Jr., March 1976. (PB-254-649)
- 107 Map Types as Aids in Using MOS PoPs in Western United States. Ira S. Brenner, August 1976. (PB-259-594)
- 108 Other Kinds of Wind Shear. Christopher D. Hill, August 1976. (PB-260-437/AS)
- 109 Forecasting North Winds in the Upper Sacramento Valley and Adjoining Forests. Christopher E. Fontana, September 1976. (PB-273-677/AS)
- 110 Cool Inflow as a Weakening Influence on Eastern Pacific Tropical Cyclones. William J. Denney, November 1976. (PB-264-655/AS)
- 112 The MAN/MOS Program. Alexander E. MacDonald, February 1977. (PB-265-941/AS)
- 113 Winter Season Minimum Temperature Formula for Bakersfield, California, Using Multiple Regression. Michael J. Oard, February 1977. (PB-273-694/AS)
- 114 Tropical Cyclone Kathleen. James R. Fors, February 1977. (PB-273-676/AS)
- 116 A Study of Wind Gusts on Lake Mead. Bradley Colman, April 1977. (PB-268-847)
- 117 The Relative Frequency of Cumulonimbus Clouds at the Nevada Test Site as a Function of K-Value. R. F. Quiring, April 1977. (PB-272-831)
- 118 Moisture Distribution Modification by Upward Vertical Motion. Ira S. Brenner, April 1977. (PB-268-740)
- 119 Relative Frequency of Occurrence of Warm Season Echo Activity as a Function of Stability Indices Computed from the Yucca Flat, Nevada, Rawinsonde. Darryl Randerson, June 1977. (PB-271-290/AS)

NOAA Technical Memorandum NWS WR-192

NWR VOICE SYNTHESIS PROJECT: PHASE I

Glen W. Sampson

Data Acquisition Division
National Weather Service Western Region
Salt Lake City, Utah
January 1986

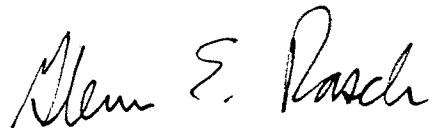
UNITED STATES
DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

National Oceanic and
Atmospheric Administration
John V. Byrne, Administrator

National Weather
Service
Richard E. Hallgren, Director



This publication has been reviewed
and is approved for publication by
Scientific Services Division,
Western Region.



Glenn E. Rasch, Chief
Scientific Services Division
Western Region Headquarters
Salt Lake City, Utah

TABLE OF CONTENTS

| | <u>PAGE</u> |
|---|-------------|
| I. Introduction | 1 |
| II. Capabilities of the System | |
| A. Purpose of the Project | 1 |
| B. How It Works | 2 |
| C. System Limitations | 3 |
| III. System Architecture | 5 |
| A. System Hardware | 5 |
| B. Software Design | 5 |
| IV. Users' Guide | 6 |
| A. System Startup Procedures | 6 |
| B. System Shutdown Procedures | 7 |
| C. If Disaster Strikes | 7 |
| D. Error Messages | 7 |
| V. Software Documentation | 8 |
| A. Radiol.c | 8 |
| B. Func.asm | 37 |
| C. Transmit.c | 47 |
| D. Disk File Descriptions and Formats | 50 |
| E. Flow Charts | 51 |
| APPENDIX A - Vocabulary List for Surface Observations | 60 |

I. INTRODUCTION

Voice synthesis technology within the last few years has made tremendous leaps and bounds in reducing the costs, and increasing the quality and capabilities of end user oriented systems. Descriptions of these new systems can be conveniently divided into two categories: text-to-voice speech (also known as synthetic speech), and phrase concatenation speech (where words and phrases are linked together to form the speech).

Synthetic speech is obviously the most useful and versatile form of the current voice synthesis technology, since any text message can be spoken regardless of format or content. A person needs only to type in what is to be spoken, and the voice system will speak it. The drawbacks to this type of system are a lower speech quality, and a relatively high mispronunciation rate of names and geographical locations (about 20%). These systems are frequently described as having a heavy "Swedish" accent.

Phrase concatenation technology on the other hand uses prerecorded words and/or phrases to produce the speech. This type of voice system has a predefined vocabulary and thus restricts the type of messages which can be spoken. Speech quality is very realistic and human sounding, with the automation aspect only apparent if the phrases or words do not link together smoothly.

Generally when people think of prerecorded phrases in the phrase concatenation technology, they think of this system being similar to a tape recorder. This idea is not true. A tape recorder records sound waves in an analog format; a voice synthesis system records speech in a digital format. Only when the voice data is in a digital format can it be efficiently controlled and manipulated by a computer.

In examining the products currently available on the market, the decision was made that synthetic voice technology does not have the high quality speech required for broadcast over NOAA Weather Radio (NWR). Therefore the system presented in this paper uses the phrase concatenation type of technology for a very high quality, versatile, voice synthesis system.

II. CAPABILITIES OF THE SYSTEM

A. Purpose of this Voice Synthesis Project

The purpose of the NWR Voice Synthesis Project is to provide a demonstration of the current voice synthesis technology. Phase I of this project is presented in this paper, and provides a complete automation of an hourly surface aviation observation for broadcast over NWR. Phase I does not include any functions other than the automation of one surface observation for one NWR console.

Future phases of this project will provide automation or semi-automation of other set format products (e.g. hourly roundups or zone forecasts). Additionally a voice override feature is planned to encompass products difficult to decode. This override feature will allow the voice synthesis system to reject an inconsistent product and ask for manual voice input.

B. How It Works

The first task in assembling a system of this type is defining the phrases or words you need in the vocabulary. The phrase concatenation voice system selected for this project provides the capabilities to both record and speak digital voice data from disk. Thus once the vocabulary is defined, a simple utility program can record your data onto the disk for future use.

The technically difficult portion now becomes writing the software to decode the raw observation data, assemble a voice product and maintain the product broadcast on NWR. Figure 1 presents the data flow on this voice synthesis system. Data is initially received from the NWS Automation of Field Operations and Services (AFOS) communication system into an IBM PC/XT. The IBM then decodes the data, encodes a voice product and maintains the product broadcast through an existing NWR console. The signals from one tape deck are interfaced to the IBM allowing the IBM to simulate and essentially replace a tape deck in the NWR console.

Figure 1 further depicts the main software modules running on the IBM PC/XT. Functional descriptions of each module are:

1. Communication routine. The communication module receives the input data from AFOS and stores these data into a circular memory buffer. Once a product is completely received, a flag is set alerting the observation decoder that a new product is ready for decoding.
2. Observation decoder routine. When a new product is received this module retrieves the product from the communication routine input buffer, and simultaneously decodes and encodes a voice product. After the observation is completely decoded, a program array containing the voice product phrases is updated for use by the broadcast routine.
3. Broadcast routine. The NWR console alerts this routine to start the observation broadcast. The proper phrases from the program array are read from the disk into memory, and spoken. Once the broadcast is finished, a start signal is transmitted to the next tape deck in the console cycle sequence.

All three of these modules are running simultaneously on the IBM to ensure no loss of data or missed broadcast sequences.

C. System Limitations

Phase I of the NWR Voice Synthesis Project in itself is a complete package, and has no limitations for decoding a single surface observation if the data is coded properly. If the data is coded wrong, the observation decoder will attempt to assemble a reasonable voice product or toss the entire observation.

Looking at this voice system as a whole and not just Phase I of this project, a number of limitations do exist in the system. The amount of vocabulary this system can maintain is limited by the amount of available disk space. Ten megabytes of disk space translates to about 40 minutes of speech. Phase I uses about 0.8 megabytes of disk space, so while this is a limitation it is not severe and can be easily overcome by increasing the disk size.

Another limitation common in all computers is the amount of available memory. An IBM PC is capable of having 640K maximum addressable memory. The features of this voice system do/could use the following amounts of memory:

| | |
|------------------------------|------|
| observation decoder | 10K |
| broadcast routine | 129K |
| user menu for interaction | 10K |
| recording routine (100 sec.) | 401K |
| TOTAL | 550K |

The Phase I software uses a slim 139K. Additional capabilities useful on the system bring the total to 550K. This leaves 90K for developing further product decoding routines. Overall the system limitations are not detrimental to expanded use in the NOAA Weather Radio program, or most other related applications.

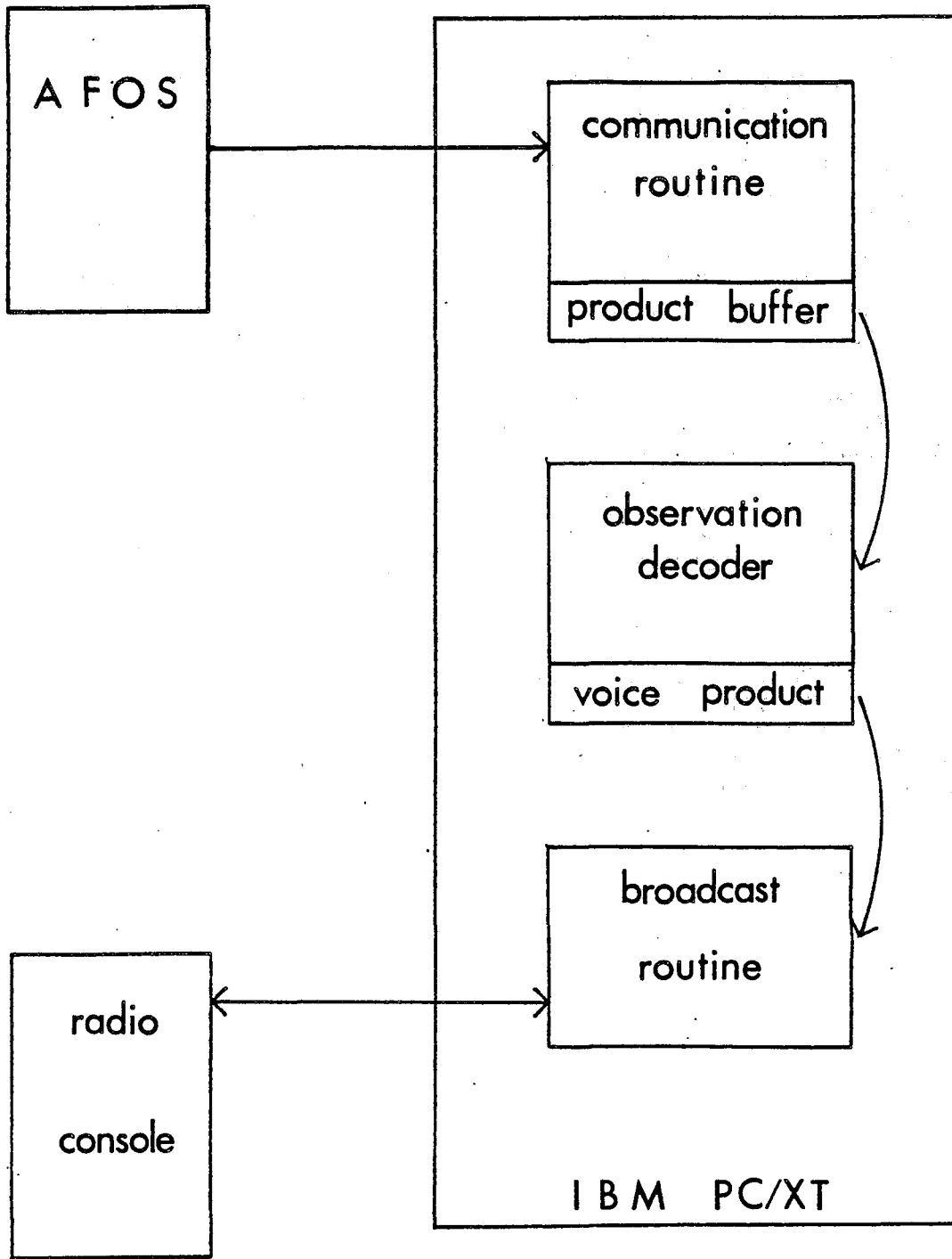


Figure 1 - Diagram depicts the data flow on the voice synthesis system, and the software modules running on the IBM.

III. SYSTEM ARCHITECTURE

A. Hardware Components

All the hardware used in this voice synthesis system consists of off-the-shelf products. The components are: a standard IBM PC/XT with monochrome monitor and keyboard, AST Six Pak Plus memory expansion, and a Vynet V101I/SD/SP voice synthesis board. The total cost of a system is approximately \$6000.

Accessories necessary for system development and helpful in general use are: a tape deck for recording your phrase speaker, a microphone, and an audio speaker. These items have a total cost of about \$300.

B. Software Design

A brief overview of the software design was presented in Figure 1 and Section IIB (How It Works). This design is closely associated with the IBM system architecture to allow a simulated multi-tasking environment. The communication routine and the broadcast routine are interrupt driven to allow the most efficient use of the system resources. Since interrupts are utilized, the operating system can remain MS-DOS which is the standard for the IBM PC. If interrupts were not utilized, the timing relationships between software modules could become a problem as the system develops. Furthermore no multi-tasking operating system for the IBM PC is currently a standard, so the software design reflects these facts.

All crucial real-time software modules (the communication and broadcast routines) are device driver level code. The decoder routine is a poll oriented routine and thus does not interfere with the real-time functions.

The communication routine handles an interrupt upon receipt of a character and puts this character into a memory buffer. When the product is completely received, a flag is set containing the location of the start of the product in the buffer. This design of the communications isolates the receipt of data from the rest of the system with the only inter-task communication conducted via the memory buffer and the product receipt flag.

The broadcast routine is very similar in design to the communication routine. The NWR console provides a signal to an asynchronous communication adapter to generate the "start of broadcast" interrupt. This interrupt triggers a transmit routine for speaking the voice product and providing a "start next tape" signal back to the NWR console when finished. After broadcast completion the regular polled routines running on the IBM are resumed. The only inter-task communication is conducted via the program array containing the voice product phrases. Future developments in this system will allow execution of the polled routines during the broadcast periods.

The surface observation decoder is triggered when the product receipt flag is set. A monitoring routine continually polls this flag until data is present. Once data is present, a check is done to determine the type

of data for starting the proper decoding/encoding function. Currently only a surface observation decoder is present, but other decoders can simply be added by writing the decoding/encoding function and including an entry in the monitoring routine.

Software maintenance and expansion of the system is relatively easy because the static functions (the communication and broadcast routines) are separated from the more dynamically oriented polled routines. Once this system is fully developed, changes should only need to occur in the polled routines. Furthermore the device drivers are the more technically difficult and hardest to change of the software routines, since they are written in assembly language. The polled routines, on the other hand, are written in a higher level language (i.e. C) enabling easier changes and access.

Additional details on the software can be found in section V (Software Documentation).

IV. USERS' GUIDE

The voice synthesis system is a prototype system and has been designed so that a user can easily determine if the system is operating properly. Three "windows" have been given the user to allow system monitoring. These "windows" are:

1. All data coming into the system are echoed on the screen with the header "Product is:". This "window" allows the user to determine if the communication task and decoder routine are working properly.

Before transmitting a product, the message appears "now transmitting" on the screen. When the broadcast is done the message appears "done transmitting". These messages allow the user to determine if the broadcast routine is working properly. Additionally the user can turn on a NWR receiver when the "now transmitting" message appears to hear the broadcast.

3. When the system is neither decoding a product nor broadcasting a product the keyboard is functional. If the user hits any keyboard key a character will be echoed to the screen. This last window insures the product receipt monitor and decoder routines are running.

If any of these "windows" fails to work the system is down and startup procedures need to be executed.

A. System Startup Procedures

The system is extremely easy to start, and can be done by two methods:

1. If the power is off, turn the power on. A red toggle power switch is located on the right rear side of the CPU box.
2. If the power is already on enter AUTOEXEC and the system will come up.

When the system has come up properly, a "System up" message will be displayed.

B. System Shutdown Procedures

System shutdown can be done by two methods depending on the circumstances:

1. If the system appears to be working properly, simultaneously type the CTRL and C keys to get a "System down" message.
2. If the "windows" indicate a system crash, turn the power off.

C. If Disaster Strikes

Since this voice synthesis system is of a prototype nature, the possibility exists for the system not to work properly. If this occurs and you can not get the system up, turn the IBM PC/XT power switch off then remove the IBM from the radio console sequencer.

D. Error Messages

When the system is being started, an initialization process occurs before the "System up" message appears. Errors can be encountered during this initialization which will abort the system from coming up. Before aborting, an error message will be displayed providing enough details for interpretation by a person familiar with the system.

Other less serious errors can occur in the surface observation decoding routine. These errors display the message "observation not decoded #", where # is a code for the type of error. These codes are:

| | |
|----|-------------------------------|
| 1 | observation not an SA or RS |
| 2 | invalid/old observation time |
| 3 | bad wx phenomena (freezing ?) |
| 4 | bad wx phenomena (ice ?) |
| 5 | bad wx phenomena (blowing ?) |
| 6 | bad wx phenomena (ground ?) |
| 7 | unknown wx phenomena |
| 8 | bad visibility (sixteenths) |
| 9 | bad visibility (eighths) |
| 10 | visibility coded improperly |
| 11 | wind coded improperly |
| 12 | air temperature beyond limits |
| 13 | dewpoint beyond limits |

If these errors occur the correction (COR) observation, which should be issued, will correct the problem and properly update the voice product.

V. SOFTWARE DOCUMENTATION

A. Radiol.c

Radiol is the main program used to generate the voice product for broadcast over the NWR console. This program initializes the system and contains monitoring routines for decoding products and processing keyboard input. The following functions are contained in this file:

| | | | |
|-----------|---------|---------|---------|
| main | loadidx | saodec | extend |
| intensity | ltg | fill | missing |
| templd | rh | vpbelo | vpabov |
| alpha | parse | unparse | loadpgm |
| fatal | strgmv | inc | dec |

Radiol uses "include" files which contain the parameter definitions, external variable declarations and the Lattice default parameters. These files are: define.c, extern.c, stdio.h and ctype.h.

```

/*      date:  May 1985
   version:  1.0
   by:      Glen W. Sampson, NWS WRH DATAC

```

Purpose:

This program is designed to automate the hourly surface aviation observation for NOAA Weather Radio. Data is received directly from AFDS, decoded and encoded into a voice product and broadcast through the NWR console.

Exits:

Errors can halt the execution of this program during the initialization phase. These error messages fit into the following categories:

1. unable to open a file;
2. unable to read from a file;
3. and, device drivers or proper hardware not installed.

Once the program is running, no fatal errors should occur.

Created by:

(Version 2.14 of Lattice C, Version 2.10 of Link)

C>lc1 radiol -mD

C>lc2 radiol -v

C>link cd radiol mface func transmit, radiol, nul, lcmd lcd

```

*/
#include "define.c"           /* get the #define parameters */
#include "extern.c"          /* ...and the extern types */
#include "stdio.h"           /* ...and the standard I/O file */
#include "ctype.h"           /* char manipulations */

main()
{
/* define the variables */
char s[20];                 /* general purpose string variable */
char c;                     /* general purpose char variable */
char inc(), dec(), getchr(); /* functions returning chars */
char *getml();              /* function returning a pointer */
int fdsao;                  /* sao voice data file descriptor */
int fd;                     /* general file descriptor */
int status;                 /* Vynet status retrieval word */
int i, j;                   /* general purpose counter */

/* initialize Vynet hardware/software */
  if (ChkDrv() != 0) {
    printf("Vynet driver is not installed\n");
    exit();
  }
  if ((status = Init(0)) == NO_HARDWARE) {
    printf("unable to initialize hardware\n");
    exit();
  }

/* catalog voice index files */
  if ((fdsao = Dopen (0, "sao_vd.idx")) == ERROR)
    fatal ("unable to open SAO_VD.IDX\n");

/* read in the data (trim the phrase number) */
  if (loadidx (&obs[0].sa_high, saophr, fdsao))

```

```

        fatal ("unable to read SAO_VD.IDX\n");
Dclose (fdsao);                               /* close file */

/* open the actual voice data file */
if ((fdsao = Dopen(0, "sao_vd.phs")) == ERROR)
    fatal ("unable to open SAO_VD.PHS\n");

/* initialize voice data buffers */
ptr1 = getm1(655361);                          /* buf #1 */
ptr2 = getm1(655361);                          /* buf #2 (64K) */

/* put headers in the data buffers */
for (i = 0; i < 7; i++) {
    ptr1[i] = header[i];
    ptr2[i] = header[i];
}

/* restore the previous program */
if ((fd = Dopen(0, "pgmfile")) == ERROR)
    fatal("unable to open PGMFILE\n");
if (Dread(fd, (unsigned) (pgmphr+1) * 8, &pgm[0]) == ERROR)
    fatal("unable to read PGMFILE\n");
for (i = 0; pgm[i].pg_fd != -1; i++);          /* find pgm end */
oldalt = pgm[i].pg_high;                       /* restore past alt */
minmax = pgm[i].pg_low;                       /* restore min/max */
Dclose(fd);                                    /* close file */

/* initialize comms port(s) and enable interrupts */
setcom(0,131);                                 /* setup the comms ports */
setcom(1,131);
setread(&buf[0],&buf[maxbuf]);              /* enable interrupts */
setstat();                                    /* int to NWR con */
prodflg = 0;                                  /* init flags */
for (i = 0; i < 25; i++)
    printf("\n");
printf("System up\n");

/* processing loop to control the voice output and decoder(s) */
ploop:
/* decode products */
if (prodflg) {                                /* product available */
    prodflg--;                                /* get array indice */
    printf("\n\nProduct is:\n\n");
    bptr = &buf[prodflg];                    /* setup buf ptr */
    for (; inc() != 3; )
        printf("%c", *bptr);
    bptr = &buf[prodflg];                    /* reset again */
    inc();                                    /* skip SOH */
    strgmv (&s, 9);                          /* trnsfr CCCNNNXXX */
    if (!strcmp (&s, "SLCSAOSLC"))
        if (c = saodec(fdsao))              /* decode product */
            printf("observation not decoded %d\n", c);
    prodflg = 0;                              /* clear prod flg */
}

/* user menu interface */
if ((i = getch()) == 3) {                    /* check for ctrl c */
    reset();
    fatal("System down\n");                  /* bye-bye */
}

```

```

        goto ploop;                                /* wait for another */
}
/*****
*
*           F U N C T I O N S
*
*****/

/***** name:  LOADIDX(IDX, NUM, FDIDX)
This routine loads an index (.IDX) file into a structure.  The phrase
number is trimmed off to save memory; therefore the phrase number must
be cataloged by the structure indice.  The number of elements in a
structure is contained in NUM.  The file "handle" is contained in FDIDX
and the structure address is contained in IDX.  Returns 1 on error.
global variables used:
    struct saoidx obs[]
routine callers:
    transmit()
*/
loadidx (idx, num, fdidx)
char idx[];
int num;
int fdidx;
{
    int i, j;
    long pos;
    char s[8];

    /* set file position, read data, move data to the struct */
    for (i = 0; i < num; i++) {
        pos = i * 8;
        j = i * 6;
        if (Dseek (fdidx, pos, 0) == ERROR) /* find position */
            return(1); /* struct offset */
        if (Dread (fdidx, (unsigned) 8, &s) == ERROR) /* set pos */
            return(1); /* error return */
        movmem (&s[2], &idx[j], 6); /* error return */
        /* fill struct */
    }
    return(0); /* normal return */
}

/***** name:  SAODEC(FDSAO)
This routine decodes a surface aviation observation and encodes a
message for broadcast on NWR.  The message is loaded into the program
file.
global variables used:
    struct pgm          ;program file
    buf[]              ;data buffer
    *bptr              ;buffer pointer
routine callers:
    main()
*/
saodec (fdsao)
int fdsao;
{
    int wx[wxmax+2];
    int vis;
    int tym;
    int i, j;
    /* sao file descriptor */
    /* current weather array */
    /* visibility number */
    /* synoptic time */
    /* general purpose counters */
}

```



```

int temp; /* temperature variable */
int temp1, temp2; /* general purpose integers */
char *ptr; /* general char ptr */
char s[10]; /* general purpose array */
char cflg; /* continue on flag */
char thinflg, niteflg; /* thin sky cvr and nite flg */

/* initialize arrays and pointers */
pgmptr = 0; /* set pgm ptr to beginning */
for (i = 0; i < wxmax + 2; i++) /* clear current wx array */
    wx[i] = 0;

/* check if obs is an 'SA' or 'RS' */
parse(lf); /* skip CCCNNNXXX */
parse(lf); /* skip WOUS/TTAA line */
parse(sp); /* skip stn id */
if (inc() != 'R' && *bptr != 'A')
    return(1); /* toss specials */
parse(sp); /* skip to time group */
if (isalpha(*bptr)) /* COR, AMD, etc? */
    parse(sp); /* skip COR, etc */

/* check time */
i = 0; /* init counter */
while (*bptr != sp) /* grab time grp */
    s[i++] = inc();
s[i] = 0; /* terminate string */
temp1 = atoi(&s); /* convert to int */
temp1 = (temp1 + 70)/100; /* calculate hour */
if (temp1 == 24) /* 00Z? */
    temp1 = 0; /* reset */
temp2 = time(&s[0]); /* get sys time */
temp2 = (temp2 + 70)/100; /* get hour again */
if (temp1 != temp2) /* compare times */
    return(2); /* bad time - error */
tym = temp2; /* save synop time */

/* start the pgm with a greeting */
temp1 = (s[0] + 30)/60 + s[1] - zofat; /* local time hour */
if (temp1 < 0) /* make adjustments */
    temp1 += 24;
else if (temp1 > 23)
    temp1 = 0;

/* put the greeting into pgm */
if (temp1 < 12) /* Good morning. */
    loadpgm(&obs[111], fdsao);
else if (temp1 > 17) /* Good evening. */
    loadpgm(&obs[113], fdsao);
else /* Good afternoon. */
    loadpgm(&obs[112], fdsao);
if (temp1 > 17 || temp1 < 9) /* dark frm 6pm-8am */
    niteflg = 1;
else /* let the sun shine */
    niteflg = 0;

/* put the obs time in pgm message */
loadpgm(&obs[114], fdsao); /* Current weather... */

/* what time is it? */

```

```

if (temp1 == 12)                                /* noon */
    loadpgm(&obs[118], fdsao);
else if (temp1 == 0)                            /* at the stroke of..*/
    loadpgm(&obs[116], fdsao);
else {                                          /* regular hour */
    if (temp1 < 12) {
        loadpgm(&obs[temp1], fdsao);
        loadpgm(&obs[117], fdsao);           /* am */
    }
    else {
        temp1 -= 12;
        loadpgm(&obs[temp1], fdsao);
        loadpgm(&obs[119], fdsao);           /* pm */
    }
}

/* sky conditions */
cflg = 1;                                       /* set the continue on flag */
while (cflg) {
    parse(sp);                                  /* get a char */
    alpha();                                    /* get a non-digit char */
    if (*bptr == '-') {
        thinflg = 1;                          /* set the thin cloud flg */
        inc();                                  /* increment pointer */
    }
    else
        thinflg = 0;                          /* clear cloud flag */
    switch (*bptr) {
        case 'C':                              /* clear */
            wx[0] = 120;
            if (niteflg)                       /* sun set already? */
                wx[0] += 1;
            cflg = 0;
            break;
        case 'E':
        case 'M':
            break;                             /* skip ceiling indicator */
        case 'S':                              /* scattered cloud deck */
            if (thinflg) {
                wx[0] = 122;                   /* mostly sunny */
                if (niteflg)                   /* sun down? */
                    wx[0] += 1;
            }
            else
                wx[0] = 124;                   /* partly cloudy */
            break;
        case 'B':                              /* broken cloud deck */
            if (thinflg)
                wx[0] = 124;                   /* partly cloudy */
            else
                wx[0] = 125;                   /* mostly cloudy */
            break;
        case 'O':                              /* overcast cloud deck */
            if (thinflg)
                wx[0] = 125;                   /* mostly cloudy */
            else
                wx[0] = 126;                   /* cloudy skies */
            break;
        case '/':
            break;                             /* all done here */
        case 'W':
            break;                             /* skip sky, get wx */
    }
}

```

```

        case 'X':
        default:
            cflg = 0;
    }
}
if (*bptr != slash)
    parse(slash);
unparse(sp);
inc();
if (isdigit(inc()) && isdigit(inc()) && isdigit(*bptr))
    unparse(sp);
unparse(sp);

/* decode visibility */
while (!isalpha(*bptr))
    dec();
parse(sp);
for (i = 0; isdigit(*bptr); inc())
    s[i++] = *bptr;
if (*bptr == sp) {
    inc();
    inc();
    if (*bptr == slash) {
        dec();
        s[i++] = *bptr;
        inc();
        inc();
        for (; isdigit(*bptr); inc())
            s[i++] = *bptr;
        s[i++] = '0';
    }
    else {
        dec();
        dec();
    }
}
else if (*bptr == slash) {
    for (inc(); isdigit(*bptr); inc())
        s[i++] = *bptr;
    s[i++] = '0';
}
s[i] = 0;
vis = atoi(&s);

/* current weather, or atmospheric phenomena */
cflg = 0;
while (cflg < wxmax)
    switch (inc()) {
        case ' ':
            cflg = wxmax;
            break;
        case 'T':
            if (extend(cflg, &wx))
                break;
            wx[cflg++] = 139;
            cflg = ltg(cflg, &wx);
            break;
        case 'R':
            if (extend(cflg, &wx))
                break;
    }
}

```

```

    if (*bptr == 'W') {                /* RW */
        wx[cflg++] = 131;
        inc();                          /* incr ptr */
    }
    else
        wx[cflg++] = 129;                /* R */
    cflg = intensity(cflg, &wx);        /* + or -? */
    break;
case 'L':                               /* drizzle */
    if (extend(cflg, &wx))
        break;
    wx[cflg++] = 144;                    /* L */
    cflg = intensity(cflg, &wx);
    break;
case 'Z':                               /* freezing precip, BRRRR */
    if (extend(cflg, &wx))
        break;
    if (*bptr == 'R')
        wx[cflg++] = 143;                /* ZR */
    else if (*bptr == 'L')
        wx[cflg++] = 145;                /* ZL */
    else
        return(3);                       /* error */
    inc();
    cflg = intensity(cflg, &wx);
    break;
case 'I':                               /* ice (fog, crystals, pelts */
    if (extend(cflg, &wx))
        break;
    switch (inc()) {
        case 'F':
            wx[cflg++] = 157;            /* IF */
            break;
        case 'C':
            wx[cflg++] = 151;            /* IC */
            break;
        case 'P':
            wx[cflg++] = 146;            /* IP */
            break;
        default:
            return(4);                   /* error */
    }
    break;
case 'S':                               /* get a shovel, its snow */
    if (extend(cflg, &wx))
        break;
    switch (inc()) {
        case 'W':
            wx[cflg++] = 148;            /* SW */
            break;
        case 'G':
            wx[cflg++] = 150;            /* SG */
            break;
        case 'P':
            wx[cflg++] = 149;            /* SP */
            break;
        default:
            wx[cflg++] = 147;            /* S */
            dec();                       /* backup */
    }
}

```

```

        cflg = intensity(cflg, &wx);      /* intensity */
        break;
    case 'B':          /* blowing stuff */
        if (extend(cflg, &wx))
            break;
        switch (inc()) {
            case 'D':
                wx[cflg++] = 156;        /* BD */
                break;
            case 'N':
                wx[cflg++] = 155;        /* BN */
                break;
            case 'S':
                wx[cflg++] = 154;        /* BS */
                break;
            case 'Y':
                wx[cflg++] = 159;        /* BY */
                break;
            default:
                return(5);                /* error */
        }
        break;
    case 'A':          /* OUCH! hail */
        if (extend(cflg, &wx))
            break;
        wx[cflg++] = 152;                /* A */
        break;
    case 'H':          /* haze */
        if (extend(cflg, &wx))
            break;
        wx[cflg++] = 135;                /* H */
        break;
    case 'K':          /* smoke */
        if (extend(cflg, &wx))
            break;
        wx[cflg++] = 133;                /* K */
        break;
    case 'D':          /* dust */
        if (extend(cflg, &wx))
            break;
        wx[cflg++] = 158;                /* D */
        break;
    case 'G':          /* ground fog */
        if (extend(cflg, &wx))
            break;
        if (*bptr == 'F')
            wx[cflg++] = 153;            /* GF */
        else
            return(6);                    /* error */
        break;
    case 'F':          /* fog */
        if (extend(cflg, &wx)) {
            j = cflg;                    /* put fog at begin */
            for (i = cflg-1; i >= 0; )
                wx[j--] = wx[i--];
            wx[0] = 128;                    /* fog and ... */
            cflg++;
        }
        else
            wx[cflg++] = 127;            /* F */

```

```

        break;
        case plus:                /* intensity symbol */
        case minus:
        break;                    /* ignore */
        default:
        return(7);                /* error processing */
    }
    loadpgm(&obs[115], fdsao);
    for (i = 0; wx[i] > 0; i++)
        loadpgm(&obs[wx[i]], fdsao);

/* decode the air temperature */
    fill(&s);                    /* get the temperature */
    if (missing(&s))             /* missing or est data */
        goto dewpt;            /* skip air temp */
    temp = atoi(&s);             /* convert to int */
    if (abs(temp) > 110)         /* within voice data limit? */
        return(12);
    temp1d (temp, 176, fdsao);   /* load temp into pgm */

/* decode the dewpoint */
dewpt:
    fill(&s);                    /* get the dewpoint */
    if (missing(&s))             /* any problems? */
        goto dfff;
    temp2 = atoi(&s);
    if (abs(temp2) > 110)         /* check limit again */
        return(13);
    temp1d (temp2, 179, fdsao); /* load dwpt into pgm */

/* decode the relative humidity */
    loadpgm(&obs[180], fdsao);    /* rh _____ */
    i = rh(temp, temp2);         /* do calculation */
    loadpgm(&obs[i], fdsao);     /* value */
    loadpgm(&obs[181], fdsao);   /* percent */

/* put the visibility into the program */
    if (vis > 20 && vis < 100)   /* good vis, skip */
        goto dfff;
    loadpgm(&obs[160], fdsao);    /* visibility, _____ */
    if (vis == 1)
        loadpgm(&obs[173], fdsao); /* one mile */
    else if (vis < 111) {
        loadpgm(&obs[vis], fdsao); /* number */
        loadpgm(&obs[161], fdsao); /* miles */
    }
    else {
        /* fraction vis */
        temp1 = vis / 1000;      /* mile or sixteenth */
        temp2 = (vis % 1000) / 100; /* numerator */
        vis = (vis % 100) / 10; /* denominator */
        switch (vis) {          /* separate denom */
            case 2:              /* halves */
                vis = 169;
                break;
            case 4:              /* quarters */
                if (temp2 == 3)   /* three quarters */
                    vis = 171;
                else              /* one quarter */
                    vis = 166;
                break;
        }
    }

```

```

case 6: /* sixteenths */
  switch (temp1) {
    case 1: /* one sixteenth */
      vis = 163;
      break;
    case 3: /* three sixteenths */
      vis = 165;
      break;
    case 5: /* five sixteenths */
      vis = 167;
      break;
    default: /* error */
      return(8);
  }
  temp1 = 0; /* reset temp1 */
  break;
case 8: /* eighths */
  switch (temp2) {
    case 1: /* one eighth */
      vis = 164;
      break;
    case 3: /* three eighths */
      vis = 168;
      break;
    case 5: /* five eighths */
      vis = 170;
      break;
    case 7: /* seven eighths */
      vis = 172;
      break;
    default:
      return(9);
  }
  break;
default: /* error */
  return(10);
}
if (temp1) { /* ...miles */
  loadpgm(&obs[173+temp1], fdsao);
  loadpgm(&obs[vis], fdsao);
  loadpgm(&obs[161], fdsao);
}
else { /* ...of a mile */
  loadpgm(&obs[vis], fdsao);
  loadpgm(&obs[162], fdsao);
}
}

```

```

/* extract the wind direction and velocity */

```

```

ddff:
  fill(&s); /* isolate the wind grp */
  if (missing(&s)) /* wnd missing */
    goto barom; /* no wnd, goto barometer */
  temp1 = atoi(&s); /* convert to integer */
  temp2 = temp1 % 100; /* speed in knots */
  temp1 = temp1 / 100; /* direction */
  if (temp2 < 2) { /* calm? */
    loadpgm(&obs[197], fdsao);
    goto barom; /* Winds are calm. */
  }
}

```

```

else if (temp2 < 7) {
    ptr = bptr;
    parse('W');
    while (inc() == 'W')
        if (inc() == 'N' && *bptr == 'D') {
            parse('V');
            while (inc() == 'V')
                if (inc() == 'R' && *bptr == 'B') {
                    loadpgm(&obs[198], fdsao);
                    bptr = ptr;
                    goto barom;
                }
        }
    bptr = ptr;
}
/* just normal wind, with possible gusts */
switch (temp1) {
case 34:
case 35:
case 36:
case 1:
case 2:
case 3:
    wx[0] = 189;
    break;
case 4:
case 5:
case 6:
    wx[0] = 190;
    break;
case 7:
case 8:
case 9:
case 10:
case 11:
case 12:
    wx[0] = 191;
    break;
case 13:
case 14:
case 15:
    wx[0] = 192;
    break;
case 16:
case 17:
case 18:
case 19:
case 20:
case 21:
    wx[0] = 193;
    break;
case 22:
case 23:
case 24:
    wx[0] = 194;
    break;
case 25:
case 26:
case 27:
case 28:

```



```

    case 29:
    case 30:
        wx[0] = 195;          /* west wind */
        break;
    case 31:
    case 32:
    case 33:
        wx[0] = 196;          /* northwest wind */
        break;
    default:
        /* error return */
        return(11);
}
loadpgm(&obs[wx[0]], fdsao); /* Winds are from the ... */
temp2 *= 1.152;             /* convert kts to mph */
loadpgm(&obs[temp2], fdsao);
for (i = 0; s[i] != 'G' && s[i] != 'Q' && s[i]; i++);
if (s[i] == 'G' || s[i] == 'Q') { /* gusting winds? */
    loadpgm(&obs[188], fdsao); /* ...mph, gusting... */
    i++; /* skip letter */
    temp2 = atoi(&s[i]) * 1.152; /* convert to mph */
    loadpgm(&obs[temp2], fdsao);
}
loadpgm(&obs[187], fdsao); /* mph */

/* decode the altimeter setting */
barom:
    parse(slash); /* get to the data */
    inc();
    for (i = 0; *bptr != slash && *bptr != sp && *bptr != etx; )
        s[i++] = inc(); /* fill the array */
    s[i] = 0; /* terminate string */
    if (missing(&s)) /* bad data? */
        goto term;
    loadpgm(&obs[199], fdsao); /* The barometer... */
    temp2 = atoi(&s); /* convert to int */
    if (s[0] < '4') { /* less than 35" */
        temp1 = 30 + s[0] - '0'; /* convert to binary */
        temp2 += 3000; /* get full value */
        loadpgm(&obs[temp1], fdsao);
    }
    else { /* greater than 24" */
        temp1 = 20 + s[0] - '0';
        temp2 += 2000; /* get full value */
        loadpgm(&obs[temp1], fdsao);
    }
    loadpgm(&obs[203], fdsao); /* ____ point ____ */
    s[1] -= '0'; /* strip the ASCII */
    s[2] -= '0';
    loadpgm(&obs[s[1]], fdsao); /* stuff # aft point */
    loadpgm(&obs[s[2]], fdsao);
    if (oldalt > temp2) /* falling */
        loadpgm(&obs[200], fdsao);
    else if (oldalt < temp2) /* rising */
        loadpgm(&obs[202], fdsao);
    else /* steady */
        loadpgm(&obs[201], fdsao);
    oldalt = temp2; /* setup prev value */

/* terminate the program format */
term:

```

```

/* first handle any supplemental data */
if (tym == 12 || tym == 0) {
    parse(etx);          /* find product end */
    unparse(slash);     /* find a slash */
    inc();              /* need a slash, space */
    while (!isspace(*bptr) && *bptr != soh) {
        unparse(slash); /* keep looking for a "/" */
        unparse(slash);
        inc();
    }
    i = 0;              /* init counter */
    cflg = 0;          /* set first time thru flg */
    while (i == 0 && *bptr != etx) {
        parse(sp);     /* skip over space */
        if (*bptr == '4' && cflg) { /* 4TTTT gp? */
            ptr = bptr; /* save pointer */
            parse(sp);  /* find end of group */
            if ((bptr - ptr) > 5) { /* max/min */
                bptr = ptr; /* restore */
                inc();      /* skip 4 */
                if (tym) { /* skip max */
                    inc();
                    inc();
                }
                s[i++] = inc(); /* grab temp */
                s[i++] = *bptr;
                break;
            }
            else /* process.. */
                bptr = ptr; /* normally */
        }
        else /* reset flg */
            cflg = 1;
        for (; isdigit(*bptr) && *bptr != etx; i++)
            s[i] = inc(); /* extract a data gp */
        s[i] = 0;
        if (i > 2) /* min/max must be 2 digits */
            i = 0; /* try again */
        else
            s[i] = 0; /* terminate */
    }
    minmax = atoi(&s[0]); /* convert to int */
    if (i == 0 || abs(minmax) > 110) { /* temp ok? */
        minmax = -9999; /* data missing flg */
        goto cont;
    }
    if (tym) { /* min */
        if (temp < minmax) /* less than zero? */
            minmax -= 100;
        templd (minmax, 182, fdsao);
    }
    else {
        if (temp > minmax) /* greater than 100 */
            minmax += 100;
        if (temp < 0 && minmax > 50)
            minmax -= 100; /* brrrr */
        templd (minmax, 183, fdsao);
    }
}

```

```

else if ((tym == 13 || tym == 14) && minmax != -9999)
    templd(minmax, 182, fdsao); /* min restored */
else if ((tym == 1 || tym == 2) && minmax != -9999)
    templd(minmax, 183, fdsao); /* max restored */
else
    minmax = -9999; /* clear minmax */

```

cont:

```

tpgm[pgmptr].tp_fd = -1; /* terminate program */
tpgm[pgmptr].tp_high = oldalt; /* save alt reading */
tpgm[pgmptr].tp_low = minmax; /* save min/max temp */

```

```

/* archive the current pgm */
temp1 = Dopen(1, "pgmfile");
if (Dwrite(temp1, (pgmchr + 1) * 8, &tpgm[0]) == ERROR)
    fatal ("unable to write to PGMFILE\n");
Dclose (temp1);

```

```

/* update the old pgm with the new one */
swappg((pgmchr+1) * 4, &tpgm[0], &pgm[0]);
return(0); /* normal return */

```

}

**** name: extend(n, w)

This routine ensures the current weather phrase (We have _____.) can be properly extended to allow for additional weather phenomena.

global variables used:

none

routine callers:

saodec()

*/

extend(n, w)

```

int n; /* the array indice */
int w[]; /* current weather array */
{
    int i; /* general purpose variable */

    if(!n) /* no need to check first */
        return(0);
    i = n - 1; /* decrement array indice */
    if (w[i] > 126 && w[i] < 141) { /* can we extend? */
        w[i] += 1; /* yes, so do it */
        return(0); /* rtn, extend ok */
    }
    else
        return(1); /* rtn, not able to */
}

```

}

**** name: intensity(n, w)

This routine attaches an intensity to the observed weather phenomena. If no intensity is found no changes are made. The array indice is returned.

global variables used:

none

routine callers:

saodec()

*/

intensity(n, w)

```

int n; /* array indice */
int w[]; /* current weather array */

```

```

int i; /* general purpose variable */

i = n - 1; /* decrement indice */
switch (*bptr) {
    case plus: /* heavy */
        w[n] = w[i]; /* move element frwd */
        w[i] = 141; /* put in intensity */
        return(n+1); /* return indice */
    case minus: /* light */
        w[n] = w[i];
        w[i] = 142;
        return(n+1);
    default: /* no intensity */
        return(n); /* no changes */
}
}

```

**** name: ltg(n, w)

This program finds the LTG contraction to determine if lightening is present during the observation of a thunderstorm.

global variables used:

```

char buf[] ;product buffer
char *bptr ;buffer pointer

```

routine callers:

```

saodec()

```

*/

ltg(n, w)

```

int n;

```

```

/* current wx array indice */

```

```

int w[];

```

```

/* current wx array */

```

```

{

```

```

char *ptr;

```

```

/* temporary buffer pointer */

```

```

ptr = bptr;

```

```

/* save the current ptr */

```

```

parse('L');

```

```

/* find an L */

```

```

while (inc() == 'L')

```

```

    if (inc() == 'T' && *bptr == 'G') {

```

```

        /* fnd LTG? */

```

```

        w[n-1] += 1;

```

```

        /* put ..and */

```

```

        w[n++] = 137;

```

```

        /* put ltg */

```

```

    }

```

```

    else

```

```

        parse('L');

```

```

        /* find L */

```

```

bptr = ptr;

```

```

/* restore pointer */

```

```

return(n);

```

```

/* return array indice */
}

```

**** name: fill(s)

This routine fills the string array (s) from buf until a slash is found.

global variables used:

```

buf[] ;buffer
bptr ;buffer pointer

```

routine callers:

```

saodec()

```

*/

fill(s)

```

char s[];

```

```

{

```

```

int i;

```

```

/* counter */

```

```

    parse(slash);                                /* get to data location */
    inc();                                        /* ... */
    for (i = 0; *bptr != slash && *bptr != etx;)
        s[i++] = inc(); /* fill the array */
    s[i] = 0;                                    /* terminate array */
    return(0);
}

/**** name: missing(s)
This routine checks to see if the input data is estimated (preceded by
an 'E'), or if the data is missing (.../MM/...). The estimated removes
the 'E' and returns 0; the missing data returns a 1. If no changes are
required this routine returns a 0.
global variables used:
    none
routine callers:
    saodec()
*/
missing(s)
char s[];
{
    int i;                                       /* general counter */

    if (s[0] == 'E')                            /* estimated data */
        for (i = 0; s[i]; i++)                /* remove 'E' */
            s[i] = s[i+1];

    if (isalpha(s[0]))                          /* any other problems? */
        return(1);                            /* error return */
    return(0);                                  /* normal return */
}

/**** name: templd(temp, n, fdsao)
This routine loads temperature data into the pgm structure.
global variables used:
    struct pgm                                ;pgm structure
routine callers:
    saodec();
*/
templd(temp, n, fdsao)
int temp;                                       /* input temperature */
int n;                                         /* temp type (air or dwpt) */
int fdsao;                                    /* file handle */
{
    loadpgm(&obs[n], fdsao);
    if (temp == 1)                             /* one degree */
        loadpgm(&obs[177], fdsao);
    else if (temp == -1)                       /* one degree below zero */
        loadpgm(&obs[184], fdsao);
    else if (temp >= 0) {                     /* _____ degrees */
        loadpgm(&obs[temp], fdsao);
        loadpgm(&obs[178], fdsao);
    }
    else {                                     /* _____ degrees below zero */
        temp = abs(temp);                     /* remove negative */
        loadpgm(&obs[temp], fdsao);
        loadpgm(&obs[185], fdsao);
    }
    return(0);
}

```

```
/***** name: rh(atem, dtem)
```

This routine calculates the relative humidity from the air and dewpoint temperatures using the Clausius-Clapeyron equation. The processed form of the equation is:

$$RH = e/es * 100\%$$

where es = the saturation vapor pressure, and
e = the vapor pressure.

Upon entry this routine must have the temperature values in Fahrenheit, and return the RH value in integer format.

reference used:

Smithsonian Meteorological Tables, Sixth Revised Edition by
Robert List; page 350.

global variables used:

none

routine callers:

saodec()

```
*/
```

```
rh(atem, dtem)
```

```
int atem, dtem;
```

```
/* air and dewpoint temps */
```

```
{
```

```
int i;
```

```
/* integer return value */
```

```
double es, e;
```

```
/* vapor pressures */
```

```
double vpabov(), vpbelo();
```

```
/* calculation functions */
```

```
float t;
```

```
/* converted temp value */
```

```
/* air temperature calculation */
```

```
t = 5./9. * (atem - 32) + 273.16;
```

```
/* Fahr to Kelvin */
```

```
if (t < 273.16)
```

```
/* below freezing? */
```

```
es = vpbelo(t);
```

```
else
```

```
es = vpabov(t);
```

```
/* above zero C */
```

```
/* dewpoint temperature calculation */
```

```
t = 5./9. * (dtem - 32) + 273.16;
```

```
/* Kelvin again */
```

```
if (t < 273.16)
```

```
e = vpbelo(t);
```

```
else
```

```
e = vpabov(t);
```

```
/* calculate rh */
```

```
i = e/es * 100;
```

```
printf("rh = %d\n", i);
```

```
return(i);
```

```
}
```

```
/***** name: vpbelo(t)
```

This routine calculates the vapor pressure for temperature below freezing, and return this value.

global variables used:

none

routine callers:

rh()

```
*/
```

```
double vpbelo(t)
```

```
double t;
```

```
{
```

```
extern double log10(), pow();
```

```
/* math functions */
```

```
float x1, x2;
```

```
/* general variables */
```

```
double e;
```

```
/* vapor pressure */
```

```

/* calculate the vapor pressure */
  x1 = 273.16 / t;          /* wrt ice */
  x2 = t / 273.16;
  e = -9.09718*(x1 - 1) - 3.56654*(log10(x1)) + 0.87679*(1 - x2);
  e = e + 0.78584;
  e = pow( (double) 10, e);      /* get the inv log */
  return(e);
}

```

```

/**** name: vpabov(t)

```

This routine calculates the vapor pressure for temperatures above freezing and return the value.

global variables used:

none

routine callers:

rh()

*/

```
double vpabov(t)
```

```
double t;
```

```
/* input temperature */
```

```
{
```

```
extern double log10(), pow();
```

```
/* math functions */
```

```
float x1, x2;
```

```
/* general variables */
```

```
double e;
```

```
/* vapor pressure */
```

```
/* calculate the vapor pressure */
```

```
x1 = 373.16 / t;
```

```
/* wrt water */
```

```
x2 = t / 373.16;
```

```
e = -7.90298 * (x1 - 1) + 5.02808 * (log10(x1)) + 3.00572;
```

```
x1 = -3.49149 * (x1 - 1);
```

```
x2 = 11.344 * (1 - x2);
```

```
x1 = pow( (double) 10, x1) - 1.0;
```

```
x2 = pow( (double) 10, x2) - 1.0;
```

```
e = e - (1.38e-7 * x2) + (8.13828e-3 * x1);
```

```
e = pow( (double) 10, e);
```

```
return(e);
```

```
}
```

```
/** end of the sa0 decoder functions **/
```

```

/***** name: ALPHA()

```

This routine finds an char which is not a digit or a space. The buffer pointer (*bptr) is left on this char upon return.

global variables used:

```
char buf[] ;product buffer
```

```
char *bptr ;buffer pointer
```

routine callers:

saodec()

*/

```
alpha()
```

```
{
```

```
while ((isdigit(*bptr) || isspace(*bptr)) && *bptr != etx)
```

```
inc(); /* increment buf pointer */
```

```
return(0); /* return to caller */
```

```
}
```

```

/***** name: PARSE(C)

```

This routine parses for a specified character (c) in buf[]. All white space characters are parsed through with the pointer left on the next valid ascii character.

global variables used:

```
char buf[] ;product buffer
```

```

char *bptr ;buffer pointer
routine callers:
saodec() fill()
*/
parse(c)
char c; /* the char to find */
{
if (c == sp) /* white space? */
while (!isspace(*bptr) && *bptr != etx)
inc(); /* find a space */
else /* specific char requested */
while (*bptr != c)
inc(); /* find the char */
while (isspace(*bptr) && *bptr != etx)
inc(); /* leave ptr on char */
return(0);
}

```

/***** name: UNPARSE(C)

This routine moves backward through the product buffer (buf[]) to find the specified char. The buffer pointer (*bptr) is left on this char.

global variables used:

```

char buf[] ;product buffer
char *bptr ;buffer pointer
routine callers:
saodec()

```

```

*/
unparse(c)
char c; /* char requested to find */
{
dec(); /* backup one char */
if (c == sp) /* white space? */
while (!isspace(*bptr) && *bptr != soh)
dec(); /* keep going */
else /* specific character */
while (*bptr != c && *bptr != soh)
dec();
return(0);
}

```

/***** name: LOADPGM (INFO, FD)

This routine fills the program structure (prepgm). Upon entry INFO contains the phrase number and FD contains the .PHS file handle. This routine increments the program pointer (pgmptr) and always returns zero.

global variables used:

```

struct prepgm
pgmptr
routine callers:
saodec() templd()

```

```

*/
loadpgm(info, fd)
int info[]; /* location of phrase data */
int fd; /* file handle of phrase data */
{
tpgm[pgmptr].tp_fd = fd; /* file descriptor */
tpgm[pgmptr].tp_high = info[0]; /* high offset */
tpgm[pgmptr].tp_low = info[1]; /* low offset */
tpgm[pgmptr].tp_length = info[2]; /* phrase length */
pgmptr++; /* inc pointer */
}

```



```

    if (pgmptr > pgmphr)
        fatal("Fgm file full, change pgmphr in define.c\n");
    return(0);
}

/***** name: FATAL(S)
This routine handles all fatal errors by displaying the string (S) and
beeping.  If this routine is called, the IBM will return to DOS and
program execution will stop!!
global variables used:
    none
routine callers:
    main          fetch
*/
fatal(s)
char *s;
{
    printf("%s", s);          /* display error msg */
    /* beep();              /* assembler beep routine */
    /*
    exit();
}

/***** name: STRGMV(S, N)
This routine fills the S char array with the specified number of chars
(N) from buf[].
global variables used:
    buf[]          ;buffer
    bptr           ;buffer pointer
routine callers:
    main()
*/
strgmv(s, n)
char s[];
int n;
{
    int i;                /* counter */
    for (i = 0; i < n; i++)
        s[i] = inc();     /* fill array */
    s[i] = 0;            /* terminate string */
    return(0);          /* return to caller */
}

/***** name: INC() and DEC()
These routine maintain the buffer pointer (*bptr) and return the
current character pointed to by *bptr.  All pointer incrementing or
decrementing must be done by these routines to avoid reading past the
circular buffer ending location.  These routines can be thought of as:

    INC() = bptr++;
    DEC() = *bptr--;
global variables used:
    buf[]          ;buffer
    bptr           ;buffer pointer
routine callers:
    saodec()       alpha()       parse()
    unparse()      fill()        ltg()
*/
char inc()
{

```

```

char c;
c = *bptr;
++bptr;
if (bptr >= &buf[maxbuf])
    bptr = &buf[0];
return(c);
}
char dec()
{
    char c;
    c = *bptr;
    --bptr;
    if (bptr < &buf[0])
        bptr = &buf[maxbuf];
    return(c);
}
/* return character */
/* grab char */
/* update pointer */
/* past buffer end? */
/* set pointer to begin */
/* return to caller */

/* same logic as inc() */

```

```
/*      date: february 1985
      version: 1.0
      by: glen w. sampson
```

extern.c

```
This file contains all of the extern (global) declarations for use by
the voice synthesis program(s).  This file partially incorporates the
Vynet file mface.inc.
```

```
*/
```

```
/* external references for Vynet driver functions */
```

```
extern int FCN_0 ();
extern int FCN_1 ();
extern int FCN_2 ();
extern int FCN_3 ();
extern int FCN_4 ();
extern int FCN_5 ();
extern int FCN_6 ();
extern int FCN_7 ();
extern int FCN_8 ();
extern int FCN_9 ();
extern int FCN_10 ();
extern int FCN_11 ();
extern int FCN_12 ();
extern int FCN_13 ();
extern int FCN_14 ();
extern int FCN_15 ();
extern int FCN_16 ();
extern int FCN_17 ();
extern int FCN_18 ();
extern int FCN_19 ();
extern int FCN_20 ();
extern int FCN_21 ();
extern int FCN_22 ();
extern int FCN_23 ();
extern int FCN_24 ();
extern int FCN_25 ();
extern int FCN_26 ();
extern int FCN_27 ();
extern int DOPEN ();
extern int DSEEK ();
extern int DREAD ();
extern int DWRITE ();
extern int DCLOSE ();
extern int SET_STAC ();
extern int CHK_DRV ();
```

```
/* structures used to manage .IDX files and program content */
```

```
/** hourly observation index catalog **/
```

```
struct saoidx {
    int sa_high;                /* hi order location */
    int sa_low;                 /* lo order location */
    unsigned int sa_length;     /* length of phrase */
} obs[saophr];                /* phrase # and quantity */
```

```
/** program of phrases **/
```

```
struct pgmidx {
    int pg_fd;                 /* phrase file number */
    int pg_high;               /* hi order phrase location */
    int pg_low;                /* low order location */
    unsigned int pg_length;    /* length of phrase */
```

```

} pgm[pgmphr + 1];

    /** program assembly structure **/
struct prepgm {
    int tp_fd;                               /* same as above struct */
    int tp_high;
    int tp_low;
    unsigned int tp_length;
} tpgm[pgmphr + 1];

/* general variables and array initialization */
char header[] = {1,0,6,0,0,0,2};           /* phrase header */
char *ptr1;                                 /* voice data buf pointer */
char *ptr2;                                 /* vd #2 buffer pointer */
char buf[maxbuf+1];                         /* raw data input buffer */
char *bptr;                                 /* buf[] pointer */
int pgmptr;                                 /* program pointer (indice) */
int oldalt;                                 /* prev obs altimeter readng */
int minmax;                                 /* min/max temp data */
int prodflg;                                /* product received flag */
int _stack = 4096;                          /* declare stack space */

```

```

/*      date:  february 1985
        version: 1.0
        by:  glen w. sampson

```

```

This file contains all of the parameter (#define) statements used in
the voice synthesis program(s).  This file incorporates portions of the
Vynet mface.inc file.
*/

```

```

/*  symbolic name to Vynet firmware function for 'C' calls  */

```

```

#define Init          FCN_0
#define OffHook       FCN_1
#define OnHook        FCN_2
#define RingOff       FCN_3
#define ChkRing       FCN_4
#define DialTone      FCN_5
#define DialPl        FCN_6
#define DialTnSt      FCN_7
#define DialPlSt      FCN_8
#define OfDlTnSt      FCN_9
#define OfDlPlSt      FCN_10
#define RdTone        FCN_11
#define RdToneSt      FCN_12
#define ChkTone       FCN_13
#define WriteVs       FCN_14
#define ReadVs        FCN_15
#define Speak         FCN_16
#define RdAlpha       FCN_17
#define RdAlpSt       FCN_18
#define Delay         FCN_19
#define InitInt       FCN_20
#define Flash         FCN_21
#define WaitDlTn      FCN_22
#define StartSd       FCN_23
#define StopSd        FCN_24
#define WaitBsAn      FCN_25
#define RdTnStEn      FCN_26
#define ChkStat       FCN_27
#define Dopen         DOPEN
#define Dseek         DSEEK
#define Dread         DREAD
#define Dwrite        DWRITE
#define Dclose        DCLOSE
#define SetStack      SET_STAC
#define ChkDrv        CHK_DRV

```

```

/*  Vynet driver status codes  */

```

```

#define SUCCESS      0x0080 /* I/O complete */
#define NO_FUNCTION  0x0081 /* invalid function number */
#define WAITING      0x0082 /* I/O pending */
#define BAD_DIGIT    0x0083 /* bad dial tone string digit */
#define TIMEOUT      0x0084 /* function timed out w/o completion */
#define OVERRUN      0x0085 /* read tone string overrun */
#define NOT_ALPHA    0x0086 /* invalid alpha code */
#define NO_HARDWARE  0x0087 /* no hardware on channel */
#define NOT_AVAILABLE 0x0088 /* function not implemented */
#define UNDERRUN     0x0089 /* speech overrun */
#define NO_INTERRUPT 0x008A /* spurious interrupt */

```

```

/*  DOS error codes  */

```

```

#define ERROR    -1

/* I/O channel numbers for voice board */
#define outchn  0          /* output channel (speaking) */
#define inchn   1          /* input channel (recording) */

/* array dimensions and initialization */
#define pgmpr   50         /* max # of phrases in pgm broadcast */
#define saophr  204        /* exact # of phrases for obs encode */
#define wxmax   3          /* # of curnt wx conds reportable */
#define zofst   7          /* hours added to local time for GMT */
#define maxbuf 1024        /* buf size, maxbuf < 64K ALWAYS!!! */

/* frequently used parameters */
#define soh     01         /* start of text */
#define etx     03         /* end of text */
#define lf      10         /* line feed */
#define cr      13         /* carriage return */
#define sp      32         /* space */
#define plus    43         /* + */
#define minus   45         /* - */
#define slash   47         /* / */

```

```

/**
 *
 * This header file defines various ASCII character manipulation macros,
 * as follows:
 *
 *      isalpha(c)      non-zero if c is alpha
 *      isupper(c)     non-zero if c is upper case
 *      islower(c)     non-zero if c is lower case
 *      isdigit(c)     non-zero if c is a digit (0 to 9)
 *      isxdigit(c)    non-zero if c is a hexadecimal digit (0 to 9, A to F,
 *                    a to f)
 *      isspace(c)     non-zero if c is white space
 *      ispunct(c)     non-zero if c is punctuation
 *      isalnum(c)     non-zero if c is alpha or digit
 *      isprint(c)     non-zero if c is printable (including blank)
 *      isgraph(c)     non-zero if c is graphic (excluding blank)
 *      iscntrl(c)     non-zero if c is control character
 *      isascii(c)     non-zero if c is ASCII
 *      iscsym(c)      non-zero if valid character for C symbols
 *      iscsymf(c)     non-zero if valid first character for C symbols
 *
 */

```

```

#define _U 1      /* upper case flag */
#define _L 2      /* lower case flag */
#define _N 4      /* number flag */
#define _S 8      /* space flag */
#define _P 16     /* punctuation flag */
#define _C 32     /* control character flag */
#define _B 64     /* blank flag */
#define _X 128    /* hexadecimal flag */

```

```
extern char _ctype[]; /* character type table */
```

```

#define isalpha(c)      (_ctype[(c)+1]&(_U|_L))
#define isupper(c)     (_ctype[(c)+1]&_U)
#define islower(c)     (_ctype[(c)+1]&_L)
#define isdigit(c)     (_ctype[(c)+1]&_N)
#define isxdigit(c)    (_ctype[(c)+1]&_X)
#define isspace(c)     (_ctype[(c)+1]&_S)
#define ispunct(c)     (_ctype[(c)+1]&_P)
#define isalnum(c)     (_ctype[(c)+1]&(_U|_L|_N))
#define isprint(c)     (_ctype[(c)+1]&(_P|_U|_L|_N|_B))
#define isgraph(c)     (_ctype[(c)+1]&(_P|_U|_L|_N))
#define iscntrl(c)     (_ctype[(c)+1]&_C)
#define isascii(c)     ((unsigned)(c)<=127)
#define iscsym(c)      (isalnum(c) || ((c)&127)==0x5f)
#define iscsymf(c)     (isalpha(c) || ((c)&127)==0x5f)

#define toupper(c)     (islower(c)?((c)-('a'-'A')):(c))
#define tolower(c)     (isupper(c)?((c)+('a'-'A')):(c))
#define toascii(c)     ((c)&127)

```

/**

*

* This header file defines the information used by the standard I/O
* package.

*

**/

```
#define _BUFSIZ 512          /* standard buffer size */
#define BUFSIZ 512         /* standard buffer size */
#define _NFILE 20          /* maximum number of files */

struct _iobuf
{
    char *_ptr;             /* current buffer pointer */
    int _rcnt;             /* current byte count for reading */
    int _wcnt;             /* current byte count for writing */
    char *_base;           /* base address of I/O buffer */
    char _flag;            /* control flags */
    char _file;            /* file number */
    int _size;             /* size of buffer */
    char _cbuf;            /* single char buffer */
    char _pad;             /* (pad to even number of bytes) */
};
```

```
extern struct _iobuf _iob[_NFILE];
```

```
#define _IOREAD 1          /* read flag */
#define _IOWRT 2          /* write flag */
#define _IONBF 4          /* non-buffered flag */
#define _IOMYBUF 8        /* private buffer flag */
#define _IOEOF 16         /* end-of-file flag */
#define _IOERR 32         /* error flag */
#define _IOSTRG 64        /* error flag */
#define _IORW 128         /* read-write (update) flag */
```

#if SPTR

#define NULL 0 /* null pointer value */

#else

#define NULL 0L

#endif

#define FILE struct _iobuf /* shorthand */

#define EOF (-1) /* end-of-file code */

#define stdin (&_iob[0]) /* standard input file pointer */

#define stdout (&_iob[1]) /* standard output file pointer */

#define stderr (&_iob[2]) /* standard error file pointer */

#define getc(p) (--(p)->_rcnt)>=0? *(p)->_ptr++:_filbf(p)

#define getchar() getc(stdin)

#define putc(c,p) (--(p)->_wcnt)>=0? ((int)(*(p)->_ptr++=(c)):_flsbf((c),p)

#define putchar(c) putc(c,stdout)

#define feof(p) ((p)->_flag&_IOEOF)!=0

#define ferror(p) ((p)->_flag&_IOERR)!=0

#define fileno(p) (p)->_file

#define rewind(fp) fseek(fp,0L,0)

#define fflush(fp) _flsbf(-1,fp)

#define clearerr(fp) clrerr(fp)

FILE *fopen();

FILE *freopen();

long ftell();


```
char *fgets();
```

```
#define abs(x) ((x)<0?-x):(x)
```

```
#define max(a,b) ((a)>(b)?(a):(b))
```

```
#define min(a,b) ((a)<=(b)?(a):(b))
```

B. Func.asm

Func.asm contains all of the assembler routines (except for mface.asm supplied by Vynet Corporation) used in the voice system. These routines include the asynchronous device drivers, and functions from the BIOS and BDOS levels of MS-DOS not found in Lattice C. The functions contained in this file are:

| | | | |
|---------|---------|--------|--------|
| time | getchr | setcom | swappg |
| setread | setstat | reset | |

The interrupt service routine addresses used in the setread and setstat functions are interrpt and talk, respectively.

```

date:  april 1985
version:  1.0
by:  glen w. sampson

```

```

; Purpose:

```

```

; These routines contain BIOS functions calls not normally available
; in C, and device drivers for the asynchronous communication adapters.

```

```

; Exits:

```

```

; No exits exist from these routines.

```

```

; Created by:

```

```

; (Version 1.0 IBM Macro Assembler)
; C>masm func, func, nul, nul

```

```

; *****
; setup the proper memory model

```

```

S_MODEL      EQU      0
P_MODEL      EQU      0
D_MODEL      EQU      1
L_MODEL      EQU      0

```

```

; *****
; setup the stack offsets

```

```

IF          L_MODEL OR P_MODEL          ; FAR CALLS
FIRST_PARAM EQU      6
SECOND_PARAM EQU     8
THIRD_PARAM EQU    10
FOURTH_PARAM EQU   12
FIFTH_PARAM EQU   14
SIXTH_PARAM EQU   16
ELSE
FIRST_PARAM EQU     4          ; NEAR CALLS
SECOND_PARAM EQU     6
THIRD_PARAM EQU     8
FOURTH_PARAM EQU   10
FIFTH_PARAM EQU   12
SIXTH_PARAM EQU   14
ENDIF

```

```

; *****
; declare external functions (if appropriate)

```

```

IF P_MODEL OR L_MODEL
EXTRN  TRANSMIT:FAR
ENDIF

```

```

; *****
; establish the DATA SEGMENT

```

```

DGROUP  GROUP  DATA
DATA    SEGMENT WORD PUBLIC 'DATA'
        ASSUME DS:DGROUP

```

```

; declare externals and variables

```

```

EXTRN  PRODFLG:WORD          ; defined in C
PTR0   DW      00H          ; buffer pointer
BEGIN0 DW      00H          ; starting prod indice
BUF0   DW      00H          ; buffer defined in C
BUF0SEG DW     00H          ; SEG of buffer
BUF0MAX DW     00H          ; end of buffer
BASE   DW      00H          ; base seg adrs

```

```
DATA EXTRA DW 00H ;extra seg reg
ENDS ;end data segment
```

```
*****
```

```
; establish the GROUP and SEGMENT definitions
```

```
IF S_MODEL
PGROUP GROUP PROG
PROG SEGMENT BYTE PUBLIC 'PROG'
ASSUME CS:PGROUP
ENDIF
```

```
IF P_MODEL
PGROUP GROUP CODE
_CODE SEGMENT BYTE PUBLIC '_CODE'
ASSUME CS:PGROUP
ENDIF
```

```
IF D_MODEL
CGROUP GROUP CODE
CODE SEGMENT BYTE PUBLIC 'CODE'
ASSUME CS:CGROUP
ENDIF
```

```
IF L_MODEL
CGROUP GROUP _PROG
_PROG SEGMENT BYTE PUBLIC '_PROG'
ASSUME CS:CGROUP
ENDIF
```

```
*****
```

```
; put the hooks in for 'C' access
```

```
PUBLIC TIME
PUBLIC GETCHR
PUBLIC SETCOM
PUBLIC SWAPPG
PUBLIC SETREAD
PUBLIC SETSTAT
PUBLIC RESET
```

```
*****
```

f u n c t i o n s

```
*****
```

```
name: time(adrs)
version: 1.0
; This routine retrieves a 2 byte value containing the hour and
; minutes. These values are stored at input address.
```

```
IF S_MODEL OR D_MODEL
TIME_PROC PROC NEAR
ELSE
TIME_PROC PROC FAR
ENDIF
```

```
TIME:
PUSH BP ;STACK'EM UP
```

```

MOV     BP,SP
PUSH   DS
MOV     AH,2CH           ; GET TIME FUNCTION CALL
INT     21H             ; DOS CAN DO IT
MOV     BX,FIRST_PARAM[BP] ; GET OFFSET
IF     L_MODEL OR D_MODEL
MOV     AX,SECOND_PARAM[BP] ; RETRIEVE SEGMENT ADRS
MOV     DS,AX           ; SETUP THE SEGMENT
ENDIF
MOV     DS:[BX],CX      ; STORE IN INPUT ADRS
MOV     AL,CH           ; PUT THE HOUR INTO 'A'
MOV     BL,64H         ; LOAD UP MULT OPERAND
MUL    BL              ; MULTIPLY BY 100
AND     CX,0FFH        ; ISOLATE THE MINUTES
ADD     AX,CX          ; COMBINE HOUR & MIN INTO INT
POP     DS             ; MOVE 'EM OUT
POP     BP
RET     ; RETURN TO CALLER

TIME_PROC ENDP

;
;   name:  getchr()
;   version:  1.0
;   This routine retrieves a character from the keyboard (with an echo)
; and returns this character to the caller.
IF     S_MODEL OR D_MODEL
GETCHR_PROC PROC NEAR
ELSE
GETCHR_PROC PROC FAR
ENDIF

GETCHR:
PUSH   BP              ; STACK 'EM
MOV     BP,SP
MOV     AH,01H         ; READ STATUS AND GET CHAR
INT     16H           ; GO FOR IT
JNZ    CLEAR          ; WE GOT IT, CLEAR OUT CHAR
XOR     AX,AX         ; NO CHAR, CLEAR AX
JMP    RTN            ; RETURN TO CALLER

CLEAR:
XOR     AH,AH         ; ZERO 'AH' TO UPDATE KB BUF
INT     16H           ; GET CHAR, CLEAR KB BUFFER
MOV     AH,0EH        ; PUT IN VIDEO OUT CODE
INT     10H           ; ECHO...echo...

RTN:
POP     BP
RET     ; ALL DONE

GETCHR_PROC ENDP

;
;   name:  setcom(port, val)
;   version:  1.0
;   This routine uses the software interrupt INT 14H to initialize an
; asynchronous comms port. Upon entry, port must be either 0 or 1
; (COM1 or COM2) and val must contain the initialization value. This
; value is documented in the BIOS source listing.
;
IF     S_MODEL OR D_MODEL
SETCOM_PROC PROC NEAR
ELSE
SETCOM_PROC PROC FAR
ENDIF

```

```

SETCOM:
    PUSH    BP                ;stack up
    MOV     BP,SP
    MOV     DX,FIRST_PARAM[BP] ;get port offset
    MOV     AX,SECOND_PARAM[BP] ;set ax for call
    INT     14H              ;set the comms port.
    POP     BP
    RET

```

```

SETCOM_PROC      ENDP

```

```

;
;      name:  swappg(count, source adrs, dest adrs)
;      version:  1.0
;      This function swaps programs and ensures this swap is uninterrupted
;      to avoid broadcasting half a new product and half an old product.
;      Upon entry, count is the number of words to move, and the addresses
;      are for the source and destination locations.

```

```

    IF S_MODEL OR D_MODEL
SWAPPG_PROC      PROC      NEAR
    ELSE
SWAPPG_PROC      PROC      FAR
    ENDIF

```

```

SWAPPG:
    PUSH    BP                ;STACK 'EM UP
    MOV     BP,SP
    PUSH    ES
    PUSH    DS
    PUSH    DI
    PUSH    SI
    MOV     CX,FIRST_PARAM[BP] ;SETUP THE COUNTER
    MOV     SI,SECOND_PARAM[BP] ;SOURCE OFFSET
    IF D_MODEL OR L_MODEL
    MOV     AX,THIRD_PARAM[BP] ;SOURCE SEGMENT
    MOV     DS,AX
    MOV     DI,FOURTH_PARAM[BP] ;DEST OFFSET
    MOV     AX,FIFTH_PARAM[BP] ;DEST SEGMENT
    MOV     ES,AX
    ELSE
    MOV     DI,THIRD_PARAM[BP] ;DEST OFFSET
    MOV     ES,DS ;FINISH ADRSING SETUP
    ENDIF
    CLI                ;DISABLE INTERRUPTS
    STI                ;REENABLE INTERRUPTS
REF      MOVSW        ;MOV THE DATA
    STI                ;REENABLE INTERRUPTS
    XOR     AX,AX      ;ZERO 'AX'
    POP     SI         ;MOVE 'EM OUT
    POP     DI
    POP     DS
    POP     ES
    POP     BP
    RET                ;RETURN TO CALLER
SWAPPG_PROC      ENDP

```

```

;
;      name:  setread(buffer beginning adrs, buffer ending adrs)
;      version:  1.0
;      This routine initializes the interrupt vector table in the IBM lower
;      memory, sets up the async. card for interrupt on character receipt,

```

```

;sets the interrupt mask on the 8259, and establishes the addresses for
;the product receipt buffer (buf[]) and product received flag
;(prodflg). Buf[] and prodflg are externs defined in 'C'.
;
        IF S_MODEL OR D_MODEL
SETREAD_PROC    PROC    NEAR
        ELSE
SETREAD_PROC    PROC    FAR
        ENDIF

; DEFINE THE INTERRUPT VECTOR AND PORT I/O ADDRESSES
        VALU    EQU    0CH                ; PRIMARY CARD TYPE
        PORT    EQU    3F8H              ; CARD I/O ADDRESS

; DEFINE THE PRODUCT HEADER AND TRAILER
        SOH     EQU    01H                ; START OF TEXT
        ETX     EQU    03H                ; END OF TEXT

SETREAD:
        PUSH    BP                        ; SAVE THE BASE
        MOV     BP,SP
        PUSH    DI
        PUSH    ES
        MOV     AX,FIRST_PARAM[BP]        ; GET OFFSET
        MOV     BUF0,AX                   ; SAVE OFFSET
        MOV     PTR0,AX                   ; INIT POINTER
        IF D_MODEL OR L_MODEL
        MOV     AX,SECOND_PARAM[BP]      ; GET THE SEGMENT
        ELSE
        MOV     AX,DS                      ; SAME AS CURRENT SEG
        ENDIF
        MOV     BUF0SEG,AX                ; SAVE THE SEG
        IF S_MODEL OR P_MODEL
        MOV     AX,SECOND_PARAM[BP]      ; GET BUFFER ENDING LOC
        ELSE
        MOV     AX,THIRD_PARAM[BP]
        ENDIF
        MOV     BUF0MAX,AX                ; SAVE THE BUF END

; INITIALIZE THE INTERRUPTS
        MOV     DX,PORT                    ; GET I/O ADRS
        ADD     DX,3                       ; CALC REGISTER ADRS
        IN     AL,DX                       ; RETRIEVE LINE STATUS
        AND     AL,07FH
        OUT    DX,AL                       ; DLAB = 0
        SUB     DX,2
        IN     AL,DX                       ; GET CURRENT INTERRUPTS
        OR     AL,01                       ; SET THE DATA AVAILABLE
        OUT    DX,AL                       ; ... INTERRUPT
        ADD     DX,3                       ; MODEM CONTROL REG
        MOV     AL,08H                     ; SETUP AL
        OUT    DX,AL                       ; CLEAR MODEM CONTROL

; INITIALIZE THE SYSTEM INTERRUPT CONTROLLER (8259) AND TABLE
        XOR     AX,AX                      ; ZERO AX
        MOV     ES,AX                      ; POINT AT INT TABLE
        MOV     DI,VALU * 4                ; INT LOCATION
        MOV     AX,OFFSET INTERRUPT       ; GET INTERRUPT OFFSET
        CLD                                ; STRING FORWARD DIR
        STOSW                              ; STORE OFFSET ADRS

```

```

MOV     AX,CS                ; GET THE SEGMENT
STOSW                ; TABLE ENTRY COMPLETE
IN     AL,21H              ; GET THE INT MASK
IF VALU EQ 0CH          ; PRIMARY PORT
AND     AL,0EFH
ELSE
AND     AL,0F7H            ; SECONDARY
ENDIF
CLI                ; DISABLE INTERRUPTS
OUT     21H,AL            ; SET ASYNC INT
STI                ; ENABLE INTERRUPTS
POP     ES
POP     DI
POP     BP
RET                ; RETURN TO CALLER

;
; INTERRUPT SERVICE ROUTINE
; REGISTERS USED ARE:
; AX - CONTAINS THE INPUT CHAR
; BX - CONTAINS THE BUFFER OFFSET (BUF[])
; CX - VARIABLE
; DX - THE I/O PORT ADDRESS
; DS - DATA SEGMENT FOR MEMORY ADRS USED IN THIS ROUTINE
; ES - DATA SEGMENT OF THE BUFFER (BUF[])
;
; INTERRUPT:
PUSH    ES                ; STACK REGISTERS USED
PUSH    DS
PUSH    AX
PUSH    BX
PUSH    CX
PUSH    DX
MOV     DX,PORT           ; SETUP PORT I/O ADRS
IN     AL,DX              ; READ CHAR
ADD     DX,5              ; LINE STATUS ADRS
PUSH    AX                ; SAVE CHAR
IN     AL,DX              ; READ LINE STATUS
AND     AL,017H           ; ISOLATE ERROR BITS
POP     AX                ; RESTORE CHAR
JZ     SKIP1              ; ERROR?
MOV     AL,'?'            ; REPLACE DATA WITH '?'
SKIP1:
MOV     BX,SEG PTR0       ; NO ERROR, CONTINUE ON
MOV     DS,BX             ; GET THE PTR SEG
MOV     BX,BUF0SEG        ; SETUP DS ADRSING
MOV     ES,BX             ; RETRIEVE BUF SEG
MOV     BX,PTR0           ; SETUP ES FOR BUF
MOV     ES:[BX],AL        ; GET CURRENT OFFSET
INC     BX                ; PUT THE CHAR IN BUF
MOV     CX,BUF0MAX        ; INCREMENT POINTER
CMP     BX,CX              ; GET THE BUF END OFFSET
JB     SKIP2              ; AT THE BUFFER END?
MOV     BX,BUF0           ; NO - CONTINUE ON
SKIP2:
MOV     PTR0,BX           ; YES - RESET PTR
CMP     AL,SOH            ; UPDATE IN MEMORY
JZ     BEGIN              ; BEGINNING OF PROD?
CMP     AL,ETX            ; END OF PROD?
JZ     FINISHED
RETURN:
CLI                ; DISABLE INTERRUPTS

```



```

MOV     AL,20H           ;CLEAR INTERRUPT
OUT     20H,AL
POP     DX
POP     CX               ;CLEAR STACK
POP     BX
POP     AX
POP     DS
POP     ES
IRET                    ;RESUME
BEGIN:
MOV     CX,BUF0         ;START OF A NEW PROD
SUB     BX,CX           ;GET THE BUF BEGINNING
MOV     BEGIN0,BX      ;CALC ARRAY ELEMENT + 1
JMP     RETURN         ;SAVE STARTING ELEMENT
FINISHED:
MOV     AX,BEGIN0      ;END OF PRODUCT
XOR     CX,CX          ;GET STARTING ELEMENT
MOV     BEGIN0,CX      ;CLEAR 'CX'
CMP     AX,00H         ;CLEAR BEGINNING INDICE
JZ      RETURN        ;VALID PROD START?
MOV     BX,SEG PRODFLG ;IF NOT, FLUSH TRASH
MOV     DS,BX          ;LOAD PRODFLG SEG
MOV     BX,OFFSET PRODFLG ;DS IS NOW PRODFLG SEG
MOV     [BX],AX       ;ACTUAL PRODFLG ADRS
JMP     RETURN        ;SET THE PRODFLG FLG...
;WITH LOC OF ARRAY + 1
SETREAD_PROC      ENDP
;
;
;   name:  setstat()
;   version:  1.0
;   This routine sets up an asynchronous card for modem status interrupt
;to trigger the speaking routine.  Interrupts are reenabled upon
;entering this routine, but all product decoding and user interaction
;is halted.
;
;   IF S_MODEL OR D_MODEL
;   EXTRN  TRANSMIT:NEAR           ;BROADCAST ROUTINE
SETSTAT_PROC  PROC  NEAR
;   ELSE
SETSTAT_PROC  PROC  FAR
;   ENDIF
; DEFINE THE INTERRUPT VECTOR AND PORT I/O ADDRESS
VALU  EQU  0BH           ; INTERRUPT VECTOR ADRS
PORT  EQU  2F8H         ; SECONDARY PORT I/O ADRS
SETSTAT:
PUSH  BP                ; STACK 'EM
MOV   BP,SP
PUSH  ES
PUSH  DI
MOV   BASE,DS          ; SAVE BASE SEG ADRS
MOV   EXTRA,ES        ; SAVE EXTRA SEG
MOV   DX,PORT + 3      ; RETRIEVE PORT ADRS
IN   AL,DX
AND   AL,07FH         ; CLEAR DLAB BIT
OUT  DX,AL             ; DLAB = 0
MOV   DX,PORT + 4      ; MODEM CONTROL ADRS
MOV   AL,0AH          ; SET RTS AND OUT2

```

```

OUT      DX,AL
MOV      DX,PORT + 1      ;LINE CONTROL ADRS
MOV      AL,08H          ;ENABLE MODEM STATUS INT
OUT      DX,AL

; SETUP THE INTERRUPT TABLE AND ENABLE SYSTEM INTERRUPTS
XOR      AX,AX            ;CLEAR AX
MOV      ES,AX           ;VECTOR TABLE SEGMENT
MOV      DI,VALU * 4     ;TABLE OFFSET
MOV      AX,OFFSET TALK  ;GET INT ROUTINE OFFSET
CLD                     ;STORE DIRECTION
STOSW                    ;OFFSET IN PLACE
MOV      AX,CS           ;GET INT ROUTINE CODE SEG
STOSW                    ;TABLE ENTRY COMPLETE
MOV      DX,PORT + 6     ;GET MODEM STATUS REGISTER
IN       AL,DX           ;CLEAR ANY POSSIBLE INT
IN       AL,21H          ;GET THE CURRENT INTERRUPTS
IF VALU EQ 00CH         ;PRIMARY CARD?
AND      AL,0EFH        ;SET INTERRUPT BIT
ELSE
AND      AL,0F7H        ;SECONDARY?
ENDIF
CLI                     ;STOP SYSTEM INTERRUPTS
OUT      21H,AL         ;ENABLE OUR INTERRUPT
STI                     ;LET THE SYSTEM RESUME
POP      DI              ;MOVE 'EM OUT
POP      ES
POP      BP
RET                     ;RETURN TO CALLER

; INTERRUPT SERVICE ROUTINE FOR SPEAKING
TALK:
PUSH     BP              ;STACK 'EM
PUSH     DS
PUSH     ES
PUSH     DI
PUSH     SI
PUSH     AX
PUSH     BX
PUSH     CX
PUSH     DX
MOV      DX,PORT + 1    ;INT ENABLE REGISTER
MOV      AL,00H        ;REMOVE INT SOURCE
OUT      DX,AL
MOV      DX,PORT + 6    ;MODEM STATUS PORT ADRS
IN       AL,DX         ;CLEAR ASYNC CARD INT
STI                     ;ENABLE INTERRUPTS
MOV      AX,SEG BASE    ;GET THE SEG OF BASE VARIABLE
MOV      DS,AX
MOV      AX,EXTRA       ;RESTORE 'ES'
MOV      ES,AX
MOV      AX,BASE        ;RESTORE 'DS'
MOV      DS,AX
CALL     TRANSMIT       ;BROADCAST THE PRODUCT
MOV      DX,PORT + 6
IN       AL,DX         ;CLEAR ANY EXTRANEIOUS INT
MOV      DX,PORT + 1    ;INT ENABLE REGISTER
MOV      AL,08H        ;MODEM STATUS INT
OUT      DX,AL         ;REENABLE ASYNC INT
CLI                     ;TURN INTERRUPTS OFF

```

```

MOV     AL,20H           ;RETURN FROM INTERRUPT
OUT     20H,AL
POP     DX               ;RETURN TO PREVIOUS TASK
POP     CX
POP     BX
POP     AX
POP     SI
POP     DI
POP     ES
POP     DS
POP     BP
IRET                    ;RESUME PREVIOUS TASK
SETSTAT_PROC      ENDP
;
;   name:  reset()
;   version:  1.0
;   This routine disables the interrupts from the async cards.

      IF S_MODEL OR D_MODEL
RESET_PROC      PROC      NEAR
      ELSE
RESET_PROC      PROC      FAR
      ENDIF

RESET:
      IN     AL,21H      ;GET THE CURRENT INT ENABLED
      OR     AL,18H      ;RESET ASYNC
      CLI                    ;DISABLE INTERRUPTS
      OUT    21H,AL      ;REMOVE ASYNC INT
      STI                    ;REENABLE SYSTEM
      XOR    AH,AH        ;CLEAR HIGH AX
      RET                    ;RETURN TO CALLER
RESET_PROC      ENDP

;
;*****
; end the appropriate segment
      IF     S_MODEL
PROG      ENDS
      ENDIF

      IF     P_MODEL
_CODE     ENDS
      ENDIF

      IF     D_MODEL
CODE      ENDS
      ENDIF

      IF     L_MODEL
_PROG     ENDS
      ENDIF

      END                    ; (END OF FILE)

```

C. Transmit.c

The transmit.c file contains two functions which maintain the broadcast on the NWR console. These routines are: fetch and transmit. Fetch moves the voice data from disk to memory, and organizes these data into the proper format. Transmit manages the memory buffers containing the voice data, "speaks" the voice phrases and starts the next tape deck on the NWR console when the voice product is finished. The transmit routines are started from talk (setstat in func.asm) via a modem signal (DTR) change generated from the NWR console.

```

/*      date:  May 1985
      version:  1.0
      by:  Glen W. Sampson

```

Purpose:

These routines maintain the broadcast to the NWR console.

Exit:

The only possible exit from this program is if an error occurs while trying to read the voice data from disk.

Created by:

```

(Version 2.14 of Lattice C)
C>lc1 transmit -mD
C>lc2 transmit -v

```

```

*/

```

```

#include "define.c"          /* parameters */
extern struct pgmidx {      /* global variables */
    int pg_fd;
    int pg_high;
    int pg_low;
    unsigned int pg_length;
} pgm[pgmphr + 1];
extern char *ptr1;
extern char *ptr2;

/***** name:  FETCH(N, PTR)
This routine fills the preallocated memory buffer with voice data,
retrieves the voice data from disk and returns the starting memory adrs
of this voice data.  Phrase N is retrieved from the program array, and
PTR contains the memory buffer address.
global variables used:
    struct pgmidx          /* program catalog */
    char header[]         /* phrase header */
routine callers:
    transmit()
*/
char *fetch(n, ptr)
int n;                    /* phr to get frm pgm catalog */
char *ptr;
{
    int status;           /* status of an operation */
    int j;                /* general purpose counter */
    long i;               /* ...ditto */

    /* setup disk and read in the voice data */
    status = Dseek(pgm[n].pg_fd, pgm[n].pg_low, pgm[n].pg_high, 0);
    status = Dread(pgm[n].pg_fd, pgm[n].pg_length, &ptr[9]);
    if (status == ERROR)
        fatal("Phrase data disk error\n");

    /* put a header and trailer on voice data */
    i = pgm[n].pg_length + 91;
    for (j = 4; j > 0; j--) /* install trailer */
        ptr[i++] = 0;
    movmem (&pgm[n].pg_length, &ptr[7], (unsigned)2);

    return(ptr);          /* rtn memory adrs */
}

```

```

/***** name: TRANSMIT()
This function is called when a request is received from the console to
broadcast a message. All other polled procedures are halted during
broadcast.
routine callers:
    talk ;interrupt service routine
global variables used:
    struct pgm ;program table
*/
transmit()
{
char *ptr; /* memory pointer from fetch() */
char *fetch(); /* function returning a ptr */
int i; /* general purpose counter */
int status; /* Vynet status retrieval word */

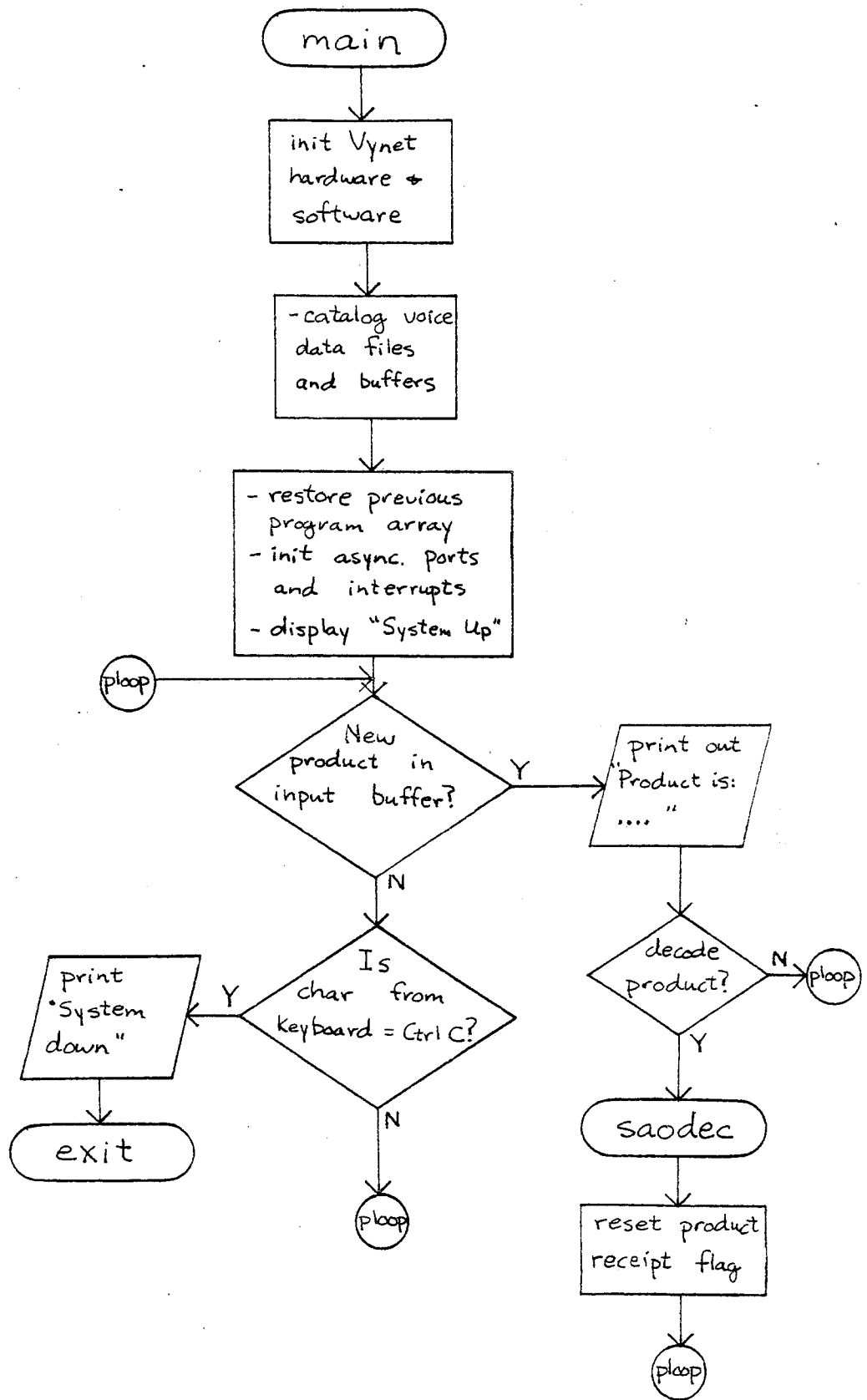
/* transmit a message */
    Delay(outchn,1); /* delay 16 milliseconds */
    printf("\nnow transmitting\n");
    ptr = fetch(0, ptr1); /* put phr in memory */
    Speak(outchn, 1, ptr); /* speak first phrase */
    for (i = 1; pgm[i].pg_fd >= 0; i++) {
        if (i % 2)
            ptr = fetch(i, ptr2); /* second buffer */
        else
            ptr = fetch(i, ptr1); /* first buffer */
        while ((status = ChkStat(outchn)) == WAITING);
        Speak(outchn, 1, ptr); /* speak next phrase */
    }
    while ((status = ChkStat(outchn)) == WAITING); /* finished? */
    printf("done transmitting\n");
    outp(764,8); /* lower the RTS */
    Delay(outchn, 10); /* delay about 1 sec */
    while ((status = ChkStat(outchn)) == WAITING); /* pause */
    outp(764,10); /* raise RTS */
    return; /* return to caller */
}

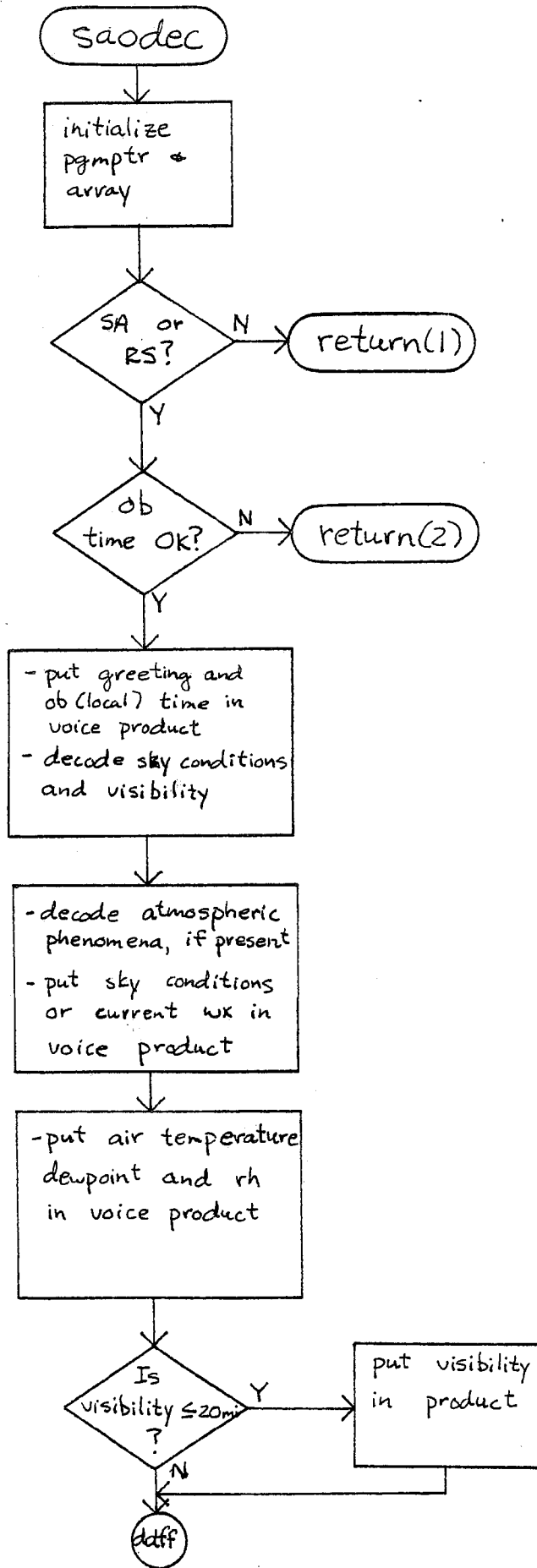
```

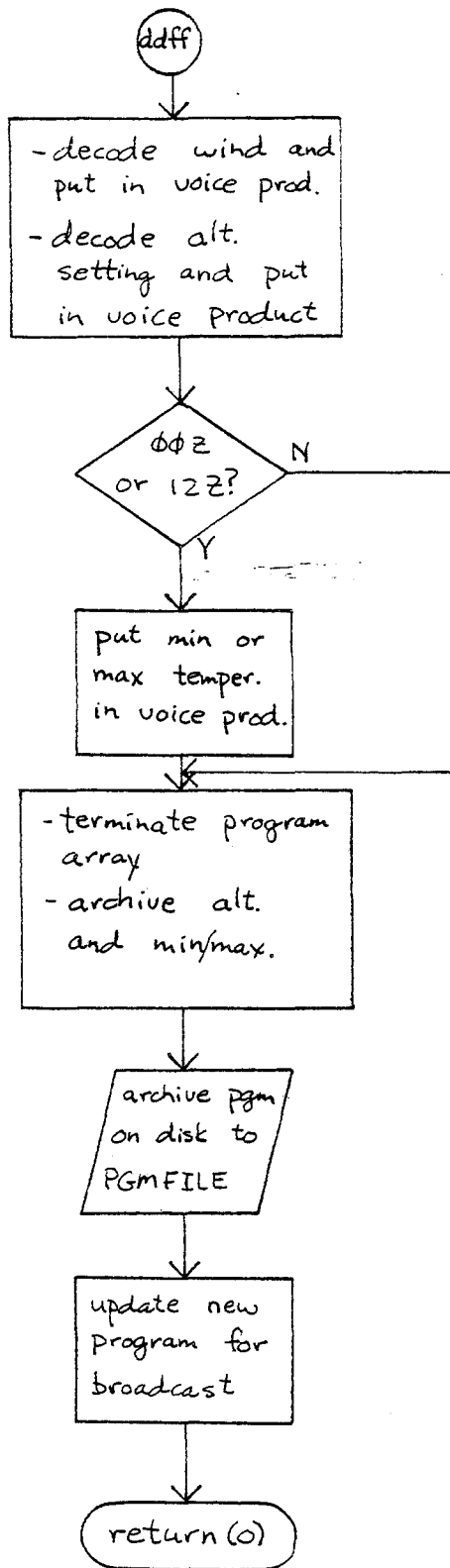
D. Disk File Descriptions and Formats

The following is a list of disk files used by the voice synthesis system with a description of its purpose and format:

1. `sao_vd.idx` - This file contains the index of voice phrases used to generate the voice product. `Sao_vd.*` is created by the Vynet utility program Digitize, which records and edits the voice input phrase onto the disk. The format is described in the Vynet User's Guide in Section 9 and is: phrase number, 2 bytes; phrase offset, 4 bytes; and, phrase length, 2 bytes.
2. `sao_vd.phs` - This file is the complement of `sao_vd.idx` and contains the actual voice data indexed by the `*.idx` file. This data is essentially the mathematical representation (ADPCM in voice synthesis lingo) of the analog voice input. Again `sao_vd.phs` is created by using the Vynet Digitize program. The format of this file is: header, 2 bytes; and up to 64534 bytes of data.
3. `pgmfile` - The last observation successfully encoded into a voice product has its phrase indexes stored in `pgmfile`. The purpose of this file is to enable restoration of the voice product if the system crashes or is taken down for a period of time. The format of `pgmfile` is: file handle of `*.phs` file, 2 bytes; phrase offset, 4 bytes; and, phrase length, 2 bytes. The file handle is a static number, since only one voice product is currently being generated. If additional voice products are generated, the file handle should become a dynamically assigned number to avoid an erroneous file handle assignment.







TIME

do a time
DOS function
call

put result
in memory
adrs ← AX

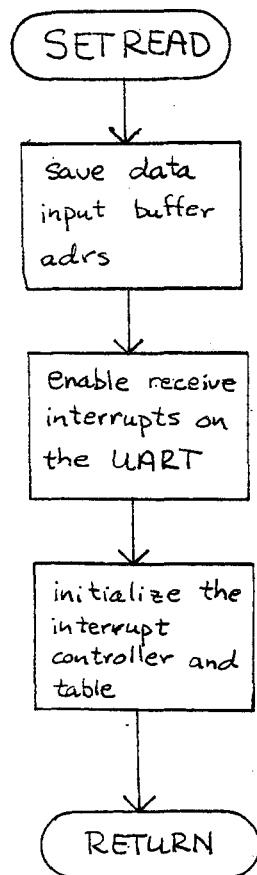
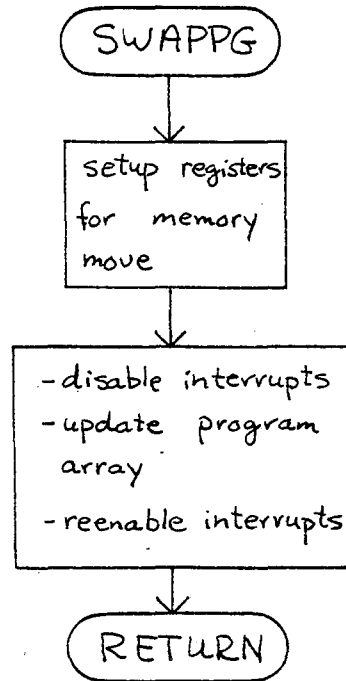
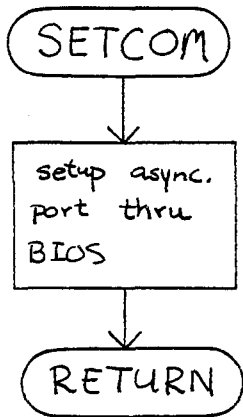
RETURN

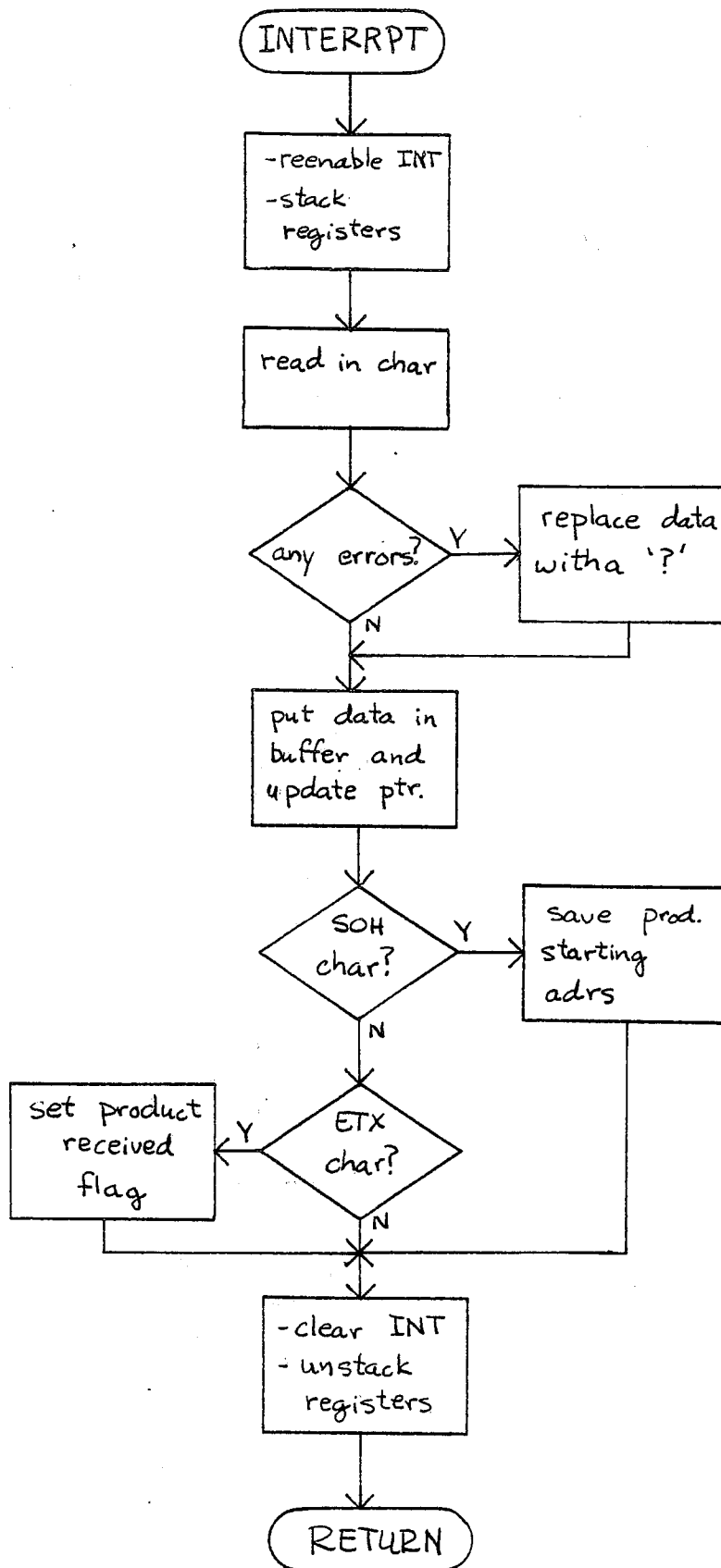
GETCHR

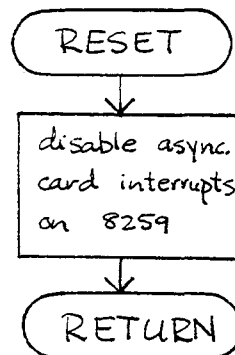
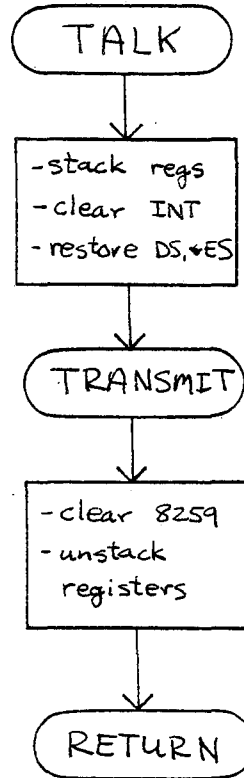
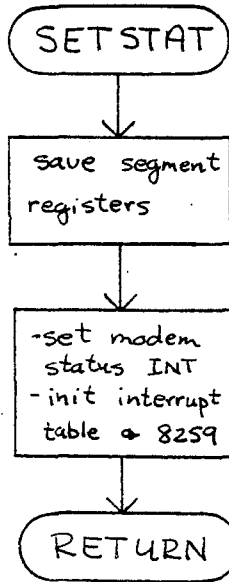
read keyboard
status & get
char thru BIOS

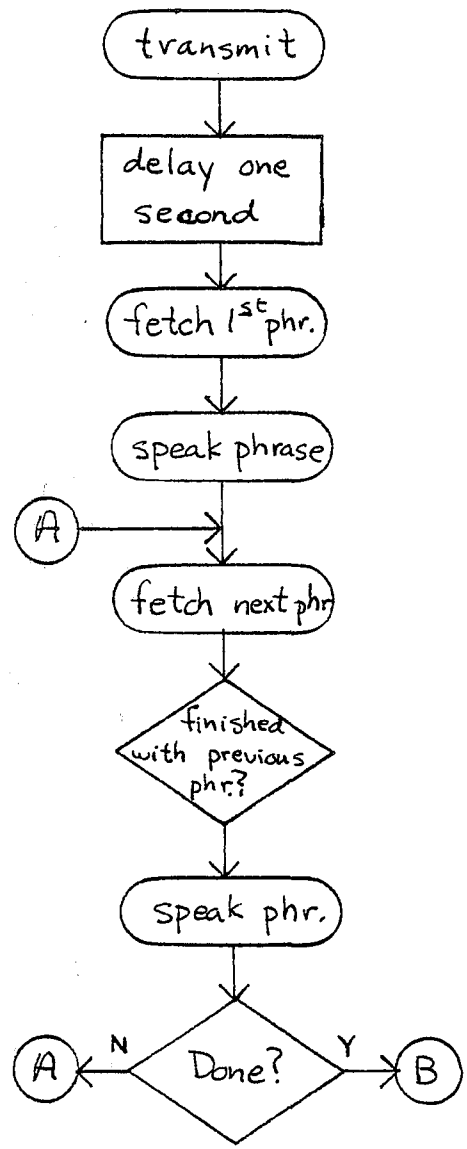
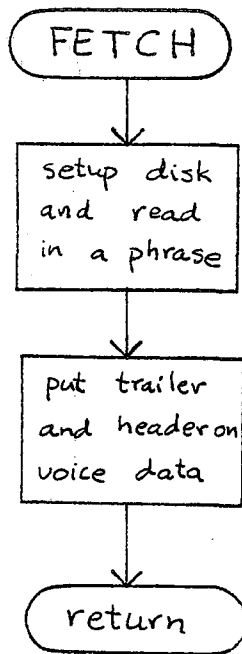
echo char
if present

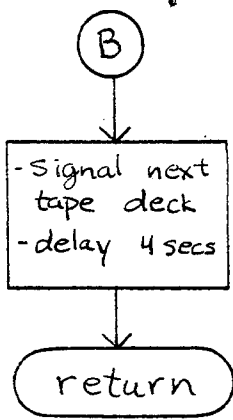
RETURN











Appendix A - Vocabulary List for Surface Observations

Phrase: Listed below.

Offset: 110

Title: greet

1. Good morning.
2. Good afternoon.
3. Good evening.

Phrase: Current weather conditions at (time) at the Salt Lake Airport. We have (current weather).

Offset: 113

Title: curwx

1. Current weather conditions at _____
2. at the Salt Lake Airport. We have _____
3. midnight
4. a.m.
5. noon
6. p.m.
7. sunny skies.
8. clear skies.
9. mostly sunny skies.
10. mostly clear skies.
11. partly cloudy skies.
12. mostly cloudy skies.
13. cloudy skies.
14. fog.
15. fog and _____.
16. rain.
17. rain and _____.
18. rain showers.
19. rain showers and _____.
20. smoke.
21. smoke and _____.
22. haze.
23. haze and _____.
24. lightening.
25. lightening and _____.
26. a thunderstorm.
27. a thunderstorm with _____.
28. heavy _____.
29. light _____.
30. freezing rain.
31. drizzle.
32. freezing drizzle.
33. ice pellets.
34. snow.
35. snow showers.
36. snow pellets.
37. snow grains.
38. ice crystals.
39. hail.
40. ground fog.
41. blowing snow.
42. blowing sand.
43. blowing dust.
44. ice fog.
45. dust.
46. blowing spray.

Phrase: Visibility, (distance) miles.

Offset: 159

Title: vis

1. Visibility, _____
2. _____ miles.
3. _____ of a mile.
4. one sixteenth (of a mile).
5. one eighth (of a mile).
6. three sixteenths (of a mile).
7. one quarter (of a mile).
8. five sixteenths (of a mile).
9. three eighths (of a mile).
10. one half (of a mile).
11. five eighths (of a mile).
12. three quarters (of a mile).
13. seven eighths (of a mile).
14. one mile.
15. one and _____.
16. two and _____.

Phrase: Temperature, (value) degrees.
Dewpoint, (value) degrees.
Relative humidity, (value) percent.

Offset: 175

Title: temp

1. Temperature, _____
2. one degree.
3. _____ degrees.
4. Dewpoint, _____
5. Relative humidity, _____
6. _____ percent.
7. The lowest temperature recorded so far this morning was _____.
8. The highest temperature recorded so far today was _____.
9. one degree below zero.
10. _____ degrees below zero.

Phrase: Winds are from the (direction at) (speed) miles an hour, gusting to (speed).

Offset: 185

Title: wind

1. (not used)
2. _____ miles an hour.
3. _____ miles and hour, gusting to _____.
4. Winds are from the north at _____
5. Winds are from the northeast at _____
6. Winds are from the east at _____
7. Winds are from the southeast at _____
8. Winds are from the south at _____
9. Winds are from the southwest at _____
10. Winds are from the west at _____
11. Winds are from the northwest at _____
12. Winds are calm.
13. Winds are light and variable.

Phrase: The barometer reads (value) point (value) inches, and is (tend-
ency).

Offset: 198

Title: pressure

1. The barometer reads _____
2. _____ inches, and is falling.
3. _____ inches, and is steady.
4. _____ inches, and is rising.
5. _____ point _____

Phrase: none (insert the numbers into phrases)

Offset: ϕ

Title: numbers

1. 0
2. 1
3. 2
4. 3
5. 4
6. 5
7. 6
8. 7
9. 8
10. 9
11. 10
12. 11
13. 12
14. 13
15. 14
16. 15
17. 16
18. 17
19. 18
20. 19
21. 20
22. 21
23. 22
24. 23
25. 24
26. 25
27. 26
28. 27
29. 28
30. 29
31. 30
32. 31
33. 32
34. 33
35. 34
36. 35
37. 36
38. 37
39. 38
40. 39
41. 40
42. 41
43. 42
44. 43
45. 44
46. 45
47. 46
48. 47
49. 48
50. 49
51. 50
52. 51
53. 52
54. 53

55. 54
56. 55
57. 56
58. 57
59. 58
60. 59
61. 60
62. 61
63. 62
64. 63
65. 64
66. 65
67. 66
68. 67
69. 68
70. 69
71. 70
72. 71
73. 72
74. 73
75. 74
76. 75
77. 76
78. 77
79. 78
80. 79
81. 80
82. 81
83. 82
84. 83
85. 84
86. 85
87. 86
88. 87
89. 88
90. 89
91. 90
92. 91
93. 92
94. 93
95. 94
96. 97
97. 96
98. 97
99. 98
100. 99
101. 100
102. 101
103. 102
104. 103
105. 104
106. 105
107. 106
108. 107
109. 108
110. 109
111. 110

NOAA Technical Memoranda NWS WR: (Continued)

- 121 Climatological Prediction of Cumulonimbus Clouds in the Vicinity of the Yucca Flat Weather Station. R. F. Quiring, June 1977. (PB-271-704/AS)
- 122 A Method for Transforming Temperature Distribution to Normality. Morris S. Webb, Jr., June 1977. (PB-271-742/AS)
- 124 Statistical Guidance for Prediction of Eastern North Pacific Tropical Cyclone Motion - Part I. Charles J. Neumann and Preston W. Leftwich, August 1977. (PB-272-661)
- 125 Statistical Guidance on the Prediction of Eastern North Pacific Tropical Cyclone Motion - Part II. Preston W. Leftwich and Charles J. Neumann, August 1977. (PB-273-155/AS)
- 127 Development of a Probability Equation for Winter-Type Precipitation Patterns in Great Falls, Montana. Kenneth B. Mielke, February 1978. (PB-281-387/AS)
- 128 Hand Calculator Program to Compute Parcel Thermal Dynamics. Dan Gudge, April 1978. (PB-283-080/AS)
- 129 Fire Whirls. David W. Goens, May 1978. (PB-283-866/AS)
- 130 Flash-Flood Procedure. Ralph C. Hatch and Gerald Williams, May 1978. (PB-286-014/AS)
- 131 Automated Fire-Weather Forecasts. Mark A. Mollner and David E. Olsen, September 1978. (PB-289-916/AS)
- 132 Estimates of the Effects of Terrain Blocking on the Los Angeles WSR-74C Weather Radar. R. G. Pappas, R. Y. Lee, B. W. Finke, October 1978. (PB289767/AS)
- 133 Spectral Techniques in Ocean Wave Forecasting. John A. Jannuzzi, October 1978. (PB291317/AS)
- 134 Solar Radiation. John A. Jannuzzi, November 1978. (PB291195/AS)
- 135 Application of a Spectrum Analyzer in Forecasting Ocean Swell in Southern California Coastal Waters. Lawrence P. Kierulff, January 1979. (PB292716/AS)
- 136 Basic Hydrologic Principles. Thomas L. Dietrich, January 1979. (PB292247/AS)
- 137 LFM 24-Hour Prediction of Eastern Pacific Cyclones Refined by Satellite Images. John R. Zimmerman and Charles P. Ruscha, Jr., Jan. 1979. (PB294324/AS)
- 138 A Simple Analysis/Diagnosis System for Real Time Evaluation of Vertical Motion. Scott Hefflick and James R. Fors, February 1979. (PB294216/AS)
- 139 Aids for Forecasting Minimum Temperature in the Wenatchee Frost District. Robert S. Robinson, April 1979. (PB298339/AS)
- 140 Influence of Cloudiness on Summertime Temperatures in the Eastern Washington Fire Weather District. James Holcomb, April 1979. (PB298674/AS)
- 141 Comparison of LFM and MFM Precipitation Guidance for Nevada During Doreen. Christopher Hill, April 1979. (PB298613/AS)
- 142 The Usefulness of Data from Mountaintop Fire Lookout Stations in Determining Atmospheric Stability. Jonathan W. Corey, April 1979. (PB298899/AS)
- 143 The Depth of the Marine Layer at San Diego as Related to Subsequent Cool Season Precipitation Episodes in Arizona. Ira S. Brenner, May 1979. (PB298817/AS)
- 144 Arizona Cool Season Climatological Surface Wind and Pressure Gradient Study. Ira S. Brenner, May 1979. (PB298900/AS)
- 145 On the Use of Solar Radiation and Temperature Models to Estimate the Snap Bean Maturity Date in the Willamette Valley. Earl M. Bates, August 1979. (PB80-160971)
- 146 The BART Experiment. Morris S. Webb, October 1979. (PB80-155112)
- 147 Occurrence and Distribution of Flash Floods in the Western Region. Thomas L. Dietrich, December 1979. (PB80-160344)
- 149 Misinterpretations of Precipitation Probability Forecasts. Allan H. Murphy, Sarah Lichtenstein, Baruch Fischhoff, and Robert L. Winkler, February 1980. (PB80-174576)
- 150 Annual Data and Verification Tabulation - Eastern and Central North Pacific Tropical Storms and Hurricanes 1979. Emil B. Gunther and Staff, EPHC, April 1980. (PB80-220486)
- 151 NMC Model Performance in the Northeast Pacific. James E. Overland, PMEL-ERL, April 1980. (PB80-196033)
- 152 Climate of Salt Lake City, Utah. Wilbur E. Figgins, October 1984. 2nd Revision. (PB85 123875)
- 153 An Automatic Lightning Detection System in Northern California. James E. Rea and Chris E. Fontana, June 1980. (PB80-225592)
- 154 Regression Equation for the Peak Wind Gust 6 to 12 Hours in Advance at Great Falls During Strong Downslope Wind Storms. Michael J. Oard, July 1980. (PB81-108367)
- 155 A Raininess Index for the Arizona Monsoon. John H. TenHarkel, July 1980. (PB81-106494)
- 156 The Effects of Terrain Distribution on Summer Thunderstorm Activity at Reno, Nevada. Christopher Dean Hill, July 1980. (PB81-102501)
- 157 An Operational Evaluation of the Scofield/Oliver Technique for Estimating Precipitation Rates from Satellite Imagery. Richard Ochoa, August 1980. (PB81-108227)
- 158 Hydrology Practicum. Thomas Dietrich, September 1980. (PB81-134033)
- 159 Tropical Cyclone Effects on California. Arnold Court, October 1980. (PB81-133779)
- 160 Eastern North Pacific Tropical Cyclone Occurrences During Intraseasonal Periods. Preston W. Leftwich and Gail M. Brown, February 1981. (PB81-205494)
- 161 Solar Radiation as a Sole Source of Energy for Photovoltaics in Las Vegas, Nevada, for July and December. Darryl Randerson, April 1981. (PB81-224503)
- 162 A Systems Approach to Real-Time Runoff Analysis with a Deterministic Rainfall-Runoff Model. Robert J. C. Burnash and R. Larry Ferral, April 1981. (PR81-224495)
- 163 A Comparison of Two Methods for Forecasting Thunderstorms at Luke Air Force Base, Arizona. Lt. Colonel Keith R. Cooley, April 1981. (PB81-225393)
- 164 An Objective Aid for Forecasting Afternoon Relative Humidity Along the Washington Cascade East Slopes. Robert S. Robinson, April 1981. (PR81-23078)
- 165 Annual Data and Verification Tabulation, Eastern North Pacific Tropical Storms and Hurricanes 1980. Emil B. Gunther and Staff, May 1981. (PB82-230336)
- 166 Preliminary Estimates of Wind Power Potential at the Nevada Test Site. Howard G. Booth, June 1981. (PB82-127036)
- 167 ARAP User's Guide. Mark Mathewson, July 1981. (revised September 1981). (PB82-196783)
- 168 Forecasting the Onset of Coastal Gales Off Washington-Oregon. John R. Zimmerman and William D. Burton, August 1981. (PB82-127051)
- 169 A Statistical-Dynamical Model for Prediction of Tropical Cyclone Motion in the Eastern North Pacific Ocean. Preston W. Leftwich, Jr., October 1981. (PR82-153883)
- 170 An Enhanced Plotter for Surface Airways Observations. Andrew J. Spry and Jeffrey L. Anderson, October 1981. (PR82-153883)
- 171 Verification of 72-Hour 500-mb Map-Type Predictions. R. F. Quiring, November 1981. (PB82-158098)
- 172 Forecasting Heavy Snow at Wenatchee, Washington. James W. Holcomb, December 1981. (PB82-177783)
- 173 Central San Joaquin Valley Type Maps. Thomas R. Crossan, December 1981. (PB82-196064)
- 174 ARAP Test Results. Mark A. Mathewson, December 1981. (PB82-193103)
- 175 Annual Data and Verification Tabulation Eastern North Pacific Tropical Storms and Hurricanes 1981. Emil B. Gunther and Staff, June 1982. (PB82-252420)
- 176 Approximations to the Peak Surface Wind Gusts from Desert Thunderstorms. Darryl Randerson, June 1982. (PB82-253089)
- 177 Climate of Phoenix, Arizona. Robert J. Schmidli, April 1969 (revised March 1983). (PB83246801)
- 178 Annual Data and Verification Tabulation, Eastern North Pacific Tropical Storms and Hurricanes 1982. E. B. Gunther, June 1983. (PB85 106078)
- 179 Stratified Maximum Temperature Relationships Between Sixteen Zone Stations in Arizona and Respective Key Stations. Ira S. Brenner, June 1983. (PB83-243904)
- 180 Standard Hydrologic Exchange Format (SHEF) Version I. Phillip A. Pasteries, Vernon C. Bissel, David G. Bennett, August, 1983. (PB85 106052)
- 181 Quantitative and Spatial Distribution of Winter Precipitation Along Utah's Wasatch Front. Lawrence B. Dunn, August, 1983. (PB85 106912)
- 182 500 Millibar Sign Frequency Teleconnection Charts - Winter. Lawrence B. Dunn, December, 1983.
- 183 500 Millibar Sign Frequency Teleconnection Charts - Spring. Lawrence B. Dunn, January, 1984. (PB85 111367)
- 184 Collection and Use of Lightning Strike Data in the Western U.S. During Summer 1983. Glenn Rasch and Mark Mathewson, February, 1984. (PB85 110574)
- 185 500 Millibar Sign Frequency Teleconnection Charts - Summer. Lawrence B. Dunn, March 1984. (PB85 111359)
- 186 Annual Data and Verification Tabulation Eastern North Pacific Tropical Storms and Hurricanes 1983. E. B. Gunther, March 1984. (PB85 109635)
- 187 500 Millibar Sign Frequency Teleconnection Charts - Fall. Lawrence B. Dunn, May 1984. (PB85 110930)
- 188 The Use and Interpretation of Isentropic Analyses. Jeffrey L. Anderson, October 1984. (PB85 132694)
- 189 Annual Data & Verification Tabulation Eastern North Pacific Tropical Storms and Hurricanes 1984. E. B. Gunther and R. L. Cross, April 1985. (PB85 187887AS)
- 190 Great Salt Lake Effect Snowfall: Some Notes and An Example. David M. Carpenter, October 1985.
- 191 NWR Voice Synthesis Project: Phase I. Glen W. Sampson, January 1986.

NOAA SCIENTIFIC AND TECHNICAL PUBLICATIONS

The National Oceanic and Atmospheric Administration was established as part of the Department of Commerce on October 3, 1970. The mission responsibilities of NOAA are to assess the socioeconomic impact of natural and technological changes in the environment and to monitor and predict the state of the solid Earth, the oceans and their living resources, the atmosphere, and the space environment of the Earth.

The major components of NOAA regularly produce various types of scientific and technical information in the following kinds of publications:

PROFESSIONAL PAPERS — Important definitive research results, major techniques, and special investigations.

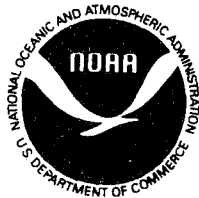
CONTRACT AND GRANT REPORTS — Reports prepared by contractors or grantees under NOAA sponsorship.

ATLAS — Presentation of analyzed data generally in the form of maps showing distribution of rainfall, chemical and physical conditions of oceans and atmosphere, distribution of fishes and marine mammals, ionospheric conditions, etc.

TECHNICAL SERVICE PUBLICATIONS — Reports containing data, observations, instructions, etc. A partial listing includes data serials; prediction and outlook periodicals; technical manuals, training papers, planning reports, and information serials; and miscellaneous technical publications.

TECHNICAL REPORTS — Journal quality with extensive details, mathematical developments, or data listings.

TECHNICAL MEMORANDUMS — Reports of preliminary, partial, or negative research or technology results, interim instructions, and the like.



Information on availability of NOAA publications can be obtained from:

**ENVIRONMENTAL SCIENCE INFORMATION CENTER (D822)
ENVIRONMENTAL DATA AND INFORMATION SERVICE
NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION
U.S. DEPARTMENT OF COMMERCE**

**6009 Executive Boulevard
Rockville, MD 20852**